Inner Product Aware Quantization: Provably Fast, Accurate, and Adaptive Algorithms

Nathan White *
University of Pennsylvania
nathanlw@cis.upenn.edu

Krish Singal *
University of Pennsylvania
ksingal@seas.upenn.edu

Abstract

Quantization is a fundamental tool used to compress datasets, neural network weights and reduce memory usage in a range of computational tasks. Many downstream applications of vector quantization perform inner products with arbitrary inputs. This motivates the study of *inner product aware* quantization schemes that approximately preserve inner products with unseen vectors, in contrast to simply minimizing the mean-squared error.

In this work, we formulate objectives that capture natural desiderata and develop *adaptive* and *unbiased* quantization methods that approximately preserve inner products with worst-case and average-case inputs. An analysis of these objectives shows a tight connection with the well-studied notion of Adaptive Stochastic Quantization (ASQ).

We develop provably fast exact and approximate algorithms for our objectives. Our theoretical results inspire efficient practical algorithms that perform well across a variety of workload distributions. They also lead to practical algorithms for standard ASQ which are 2–10$\times$ faster than prior state-of-the-art methods while maintaining quality. These theoretical and empirical results contribute towards making adaptive quantization techniques more efficient and tractable in practical settings.

1 Introduction

Vector quantization is an incredibly important tool that is central to space and runtime optimization in a wide variety of computational and machine learning applications. For example, quantization is used in dataset compression for vector search [GHKS13, GGX+25], compression of large model weights and key-value caches [SZY+23], quantization-aware training [MNA+17], and post-training quantization [FAHA23, JLPK23].

Formally, we define vector quantization as follows. Consider $w \in \mathbb{R}^d$ and let $Q \subset \mathbb{R}$ be a set of *quantization values* such that $|Q| = s \ll d$. Then, the vector w can be *quantized* via some *rounding distribution* $\mathcal{D}$ with $\text{Supp}(\mathcal{D}) \subseteq Q^d$. In this work, we consider *unbiased* and *adaptive* quantization schemes. A quantization scheme is unbiased if $\mathbf{E}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}}[\widehat{\boldsymbol{w}}] = w$,[2] while an adaptive scheme is one where both Q and $\mathcal{D}$ may depend on w. In full generality, we wish to jointly optimize over the tuple $(Q, \mathcal{D})$ according to some suitable objective function.

Much prior work on quantization considers schemes which are either non-adaptive/weakly adaptive or biased; for example, schemes such as QSGD [AGL+17] and NUQSGD [RKFM+21] only use superficial properties of the vectors such as their norm or aspect ratio. Other works [FHH+20, FTM+20] are tailored to vectors which come from common, pre-specified distributions, but do not

*Equal Contribution
[2]We use boldface to denote random variables.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

adapt to the specific input vector. For example, there is evidence that important and relevant data in ML applications follow LogNormal [CBUS+20], Normal [BNS19], or sub-Weibull distributions [VAM18]. On the other hand, techniques such as Round-to-Nearest (RTN), which are biased and non-adaptive, are known to be outperformed by adaptive methods on post-training quantization [NAVB+20].

A line of work on *adaptive stochastic quantization* (ASQ) in [BBBIMV24, ZLK+17, FTM+20] studies adaptive and unbiased quantization schemes, and notes substantial improvements over other types of quantization (see Figure 1 of [BBBIMV24]). In ASQ, the goal is to construct a quantization set which minimizes the mean-squared error

$$\mathsf{MSE}(\widehat{\boldsymbol{w}}, w) \triangleq \mathbf{E}\left[\|\widehat{\boldsymbol{w}} - w\|_2^2\right].$$

The rounding distribution from which $\widehat{\boldsymbol{w}}$ is sampled is simple and natural: round each w_i independently to either the closest quantization point larger than w_i or smaller than w_i in the unique unbiased manner. We refer to this rounding distribution as *standard stochastic quantization*, and denote the resulting distribution $\mathcal{D}^{\mathsf{SSQ}}(w, Q)$;[3] we give a formal definition in Section 1.1. Standard stochastic quantization is a natural rounding distribution choice, as for each coordinate i, it minimizes $\mathbf{Var}[\widehat{\boldsymbol{w}}_i]$ across all unbiased distributions.[4]

While MSE is a natural objective to optimize, many downstream applications of quantization rely on more than simply low ℓ_2 distance. For instance, many modern applications take inner products of quantized vectors with other, potentially arbitrary vectors (as is the case in both neural network weight quantization and vector search). Thus, the success of quantization in these applications heavily depends on its ability to preserve inner products well.

In this work, we introduce and study notions of *inner product aware* quantization objectives which aim to approximately preserve inner products. Because we consider unbiased quantization schemes where $\mathbf{E}[\widehat{\boldsymbol{w}}] = w$, linearity of expectation gives that for any vector $x \in \mathbb{R}^d$, $\mathbf{E}[\langle \widehat{\boldsymbol{w}}, x\rangle] = \langle w, x\rangle$. So, minimizing the expected (squared) error of $\langle \widehat{\boldsymbol{w}}, x\rangle$ reduces to minimizing $\mathbf{Var}[\langle \widehat{\boldsymbol{w}}, x\rangle]$. In our first objective, we minimize the variance over the worst-case choice of x.

Definition 1.1 (Maximum Directional Variance (MDV))**.** For a vector $w \in \mathbb{R}^d$ and target quantization set size $s \in \mathbb{N}$, we define

$$\mathsf{MDV}(w, s) \triangleq \min_{\substack{Q \subset \mathbb{R} \\ |Q| \le s}} \max_{\substack{x \in \mathbb{R}^d \\ \|x\|_2 \le 1}} \mathop{\mathbf{Var}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w, Q)} [\langle \widehat{\boldsymbol{w}}, x\rangle]$$

In practice, input vectors x are often not worst-case, and instead come from some known (or estimated) distribution. As such, we also consider optimizing to minimize the average variance over some (known) distribution of vectors x.

Definition 1.2 (Average Directional Variance (ADV))**.** For a vector $w \in \mathbb{R}^d$, target quantization set size $s \in \mathbb{N}$, and input distribution $\mathcal{X}$ over $\mathbb{R}^d$, define

$$\mathsf{ADV}_{\mathcal{X}}(w, s) \triangleq \min_{\substack{Q \subset \mathbb{R} \\ |Q| \le s}} \mathop{\mathbf{E}}_{\boldsymbol{x} \sim \mathcal{X}} \left[\mathop{\mathbf{Var}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w, Q)} \left[\langle \widehat{\boldsymbol{w}}, \boldsymbol{x}\rangle\right] \right] \tag{1}$$

We note that these objectives fix standard stochastic quantization as the rounding distribution and optimize over quantization sets. While this is a natural and standard distribution choice, one may wonder if a better choice of rounding distribution could lead to lower inner product variance. We show that this is unlikely to be a fruitful endeavor: for MDV, we prove that standard stochastic quantization is in fact the optimal rounding distribution (Theorem 1) and for ADV we show that it is NP-Hard to compute the optimal distribution (Theorem 2).

For both of these objectives, we design algorithms which are provably fast and obtain a solution within a $(1 + \varepsilon)$ factor of the optimal objective cost, for a user-specified $\varepsilon > 0$. See Section 3 for a discussion of our algorithmic results.

We also implement and evaluate practical versions of these algorithms, and show they outperform previous algorithms for adaptive quantization. In particular, due to a close connection between

[3] When clear from context, we simply write $\mathcal{D}^{\mathsf{SSQ}}$.

[4] See Theorem 1 for a proof of this fact.

ADV and traditional ASQ (see Section 1.1), we employ our techniques to develop algorithms that significantly outperform the current state-of-the-art `QUIVER` algorithm [BBBIMV24]; see Figure 1. In all plots and tables, we denote our novel algorithms with an asterisk*.

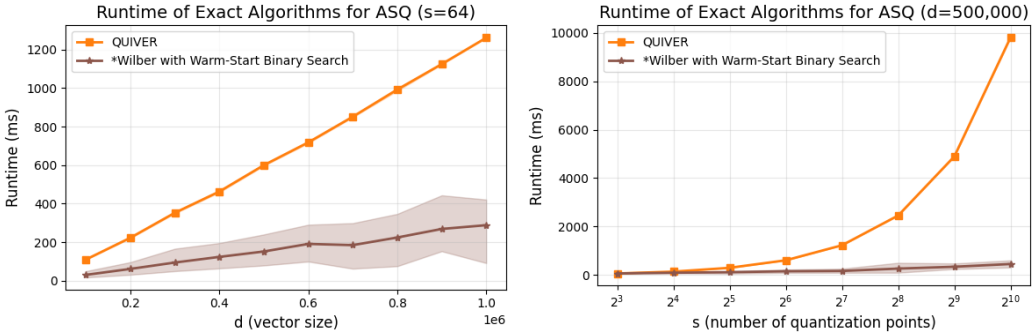


Figure 1: Figure comparing the runtime of the previous state-of-the-art algorithm for traditional ASQ with our faster algorithm (`Wilber` with Warm-Start Binary Search), on vectors drawn from $\text{LogNormal}(0,1)$. On the left, we fix $s = 64$ and vary the vector size d, while on the right we fix $d = 500{,}000$ and vary s. The shaded regions show the 10-90% range of runtimes across sampled vectors. Figure 3 is similar, with additional methods compared as well.

Since the exact algorithm is a core subroutine of the approximate `QUIVER` algorithm of [BBBIMV24], we immediately obtain speedups for this as well. More generally, any downstream task using `QUIVER` as a subroutine (e.g., [BBBIMV25]) can be sped up using our implementation. See Section 4 for the details of our empirically fast algorithms.

Because adaptive methods come with an inherent trade-off (higher quality quantization at the cost of slower pre-processing time), one of the main impacts of our work is the use of theoretical insights to advance adaptive techniques towards efficiency in practical settings.

1.1 Preliminaries

Standard Stochastic Quantization. For a given vector $w \in \mathbb{R}^d$ and quantization set Q, define the *standard stochastic quantization* rounding distribution $\mathcal{D}^{\mathsf{SSQ}}(w, Q)$ to be the distribution which independently rounds each w_i such that for $\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w, Q)$,

$$\widehat{\boldsymbol{w}}_i = \begin{cases} w_i^{\uparrow} & \text{w.p. } (w_i - w_i^{\downarrow})/(w_i^{\uparrow} - w_i^{\downarrow}) \\ w_i^{\downarrow} & \text{w.p. } (w_i^{\uparrow} - w_i)/(w_i^{\uparrow} - w_i^{\downarrow}) \end{cases}$$

where $w_i^{\uparrow} := \min\{q \in Q : q \geq w_i\}$ and $w_i^{\downarrow} := \max\{q \in Q : q \leq w_i\}$ are the values of Q closest to w_i from above and below, respectively. Note that $\mathbf{E}[\widehat{\boldsymbol{w}}] = w$ and $\mathbf{Var}[\widehat{\boldsymbol{w}}_i] = (w_i^{\uparrow} - w_i)(w_i - w_i^{\downarrow})$ for all $i \in [d]$. For notational ease, we often write $\mathsf{VarSSQ}(w_i, Q) := \mathbf{Var}[\widehat{\boldsymbol{w}}_i] = (w_i^{\uparrow} - w_i)(w_i - w_i^{\downarrow})$.

Properties of MDV and ADV. Here, we make some simple but useful observations about the structure of the MDV and ADV objectives. We first introduce notational shorthands for the costs of each objective for a fixed quantization set. We write

$$\mathsf{MDV}(w, Q) \triangleq \max_{\substack{x \in \mathbb{R}^d \\ \|x\|_2 \leq 1}} \mathbf{Var}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w,Q)}[\langle \widehat{\boldsymbol{w}}, x \rangle] \quad \text{and} \quad \mathsf{ADV}_{\mathcal{X}}(w, Q) \triangleq \mathop{\mathbf{E}}_{\boldsymbol{x} \sim \mathcal{X}} \left[\mathbf{Var}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w,Q)}[\langle \widehat{\boldsymbol{w}}, \boldsymbol{x} \rangle] \right]$$

MDV has a simple combinatorial structure; namely,

$$\begin{aligned} \mathsf{MDV}(w, Q) &= \max_{\substack{x \in \mathbb{R}^d \\ \|x\|_2 \leq 1}} \left[\mathbf{Var}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w,Q)}[\langle \widehat{\boldsymbol{w}}, x \rangle] \right] \\ &= \max_{\substack{x \in \mathbb{R}^d \\ \|x\|_2 \leq 1}} \left[\sum_{i=1}^{d} x_i^2 \mathbf{Var}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w,Q)}[\widehat{\boldsymbol{w}}_i] \right] \qquad \text{(Independence of } \mathcal{D}^{\mathsf{SSQ}}\text{)} \end{aligned}$$

$$= \max_{j \in [d]} \mathsf{VarSSQ}(w_j, Q).$$

So, the worst-case input vector x is the standard unit basis vector e_i where i is such that $\mathsf{VarSSQ}[w_i]$ is maximized. For ADV, consider a distribution $\mathcal{X}$ supported on $\mathbb{R}^d$ and let $\lambda_i \triangleq \mathbf{E}_{\boldsymbol{x} \sim \mathcal{X}}[\boldsymbol{x}_i^2]$. Then,

$$\mathsf{ADV}_{\mathcal{X}}(w, s) = \min_{\substack{Q \subset \mathbb{R} \\ |Q| \le s}} \mathop{\mathbf{E}}_{\boldsymbol{x} \sim \mathcal{X}} \left[\mathop{\mathbf{Var}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w, Q)} \left[\langle \widehat{\boldsymbol{w}}, \boldsymbol{x} \rangle \right] \right] = \min_{\substack{Q \subset \mathbb{R} \\ |Q| \le s}} \mathop{\mathbf{E}}_{\boldsymbol{x} \sim \mathcal{X}} \left[\sum_{i \in [d]} \boldsymbol{x}_i^2 \cdot \mathsf{VarSSQ}[w_i] \right]$$
$$= \min_{\substack{Q \subset \mathbb{R} \\ |Q| \le s}} \sum_{i \in [d]} \lambda_i \cdot \mathsf{VarSSQ}[w_i]$$

which is exactly the weighted MSE. This connection between quantization for average-case inner product preservation and that of the (weighted) MSE objective also provides one possible theoretical explanation for the empirical success of Adaptive Stochastic Quantization for downstream applications in ML and vector search.

2 Background and Motivation

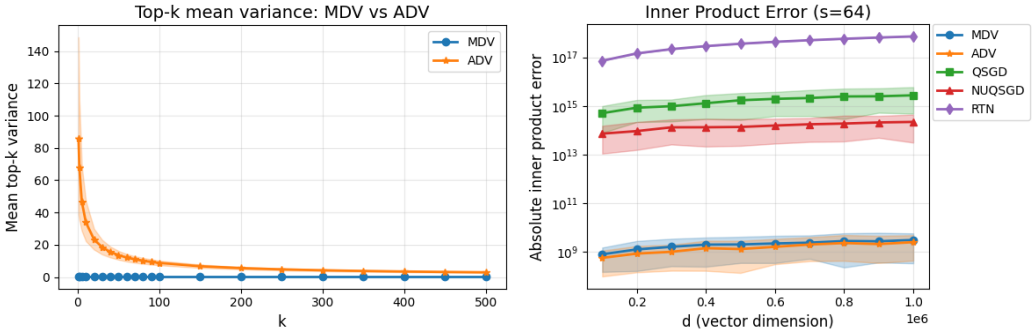


Figure 2: (a) Average variance $\mathsf{VarSSQ}[w_i]$ of the worst k coordinates on vectors w drawn from LogNormal(0,1) over 100 trials. (b) Average absolute inner product error on vectors w drawn from a mixture of 16 Gaussians with variance 10 and means separated by 10^5. In both plots, the shaded regions show the 10–90% range across trials.

We now briefly discuss the merits of unbiased and adaptive quantization methods. Methods such as NUQSGD and QSGD [RKFM+21, AGL+17] are unbiased but only very weakly adaptive; they choose the quantization set Q only using information about the range of the entries in w. Round-to-Nearest is both non-adaptive and biased, and there is evidence [NAVB+20] that post-training quantization using RTN is significantly outperformed by adaptive techniques. We note that [ZDHM25] study quantization under the objective of worst-case inner product preservation. However, they work in the non-adaptive setting, where the quantization set cannot depend on the vector being quantized.

Figure 1 in [BBBIMV24] illustrates the superiority of unbiased and adaptive techniques when minimizing the mean-squared error. As noted in the introduction, the worst-case variance $\mathbf{Var}[\langle \widehat{\boldsymbol{w}}, x \rangle]$ is realized by the standard basis vector e_i where i is such that $\mathsf{VarSSQ}[w_i]$ is largest. Figure 2(a) shows that, empirically, optimizing MDV actually minimizes the worst variances beyond the single worst coordinate, when compared against a solution to ADV. Indeed, on at least the 500 worst coordinates of a 1,000,000-dimensional vector (i.e. the worst 0.05%), the optimal quantization set for MDV has smaller variance than that for ADV. Figure 2(b) further gives evidence for the benefits of using unbiased and adaptive methods for worst-case inner product preservation.

In Table 4 (see Appendix H), we present preliminary evidence that our methods have improved tail performance over the popular technique of product quantization (PQ) for certain types of vector search (namely maximum inner product search). While ADV obtains recall comparable to PQ across both the average and tail events for ℓ_2 search, it achieves significantly higher recall on the worst 0.1% of queries tested for maximum inner product search. These results, while limited in scope, suggest that explicitly optimizing for inner product preservation results in better quantization performance, as PQ is not tailored to inner product preservation.

3 Theoretical Results

3.1 Optimizing the Rounding Distribution

Our main objectives MDV and ADV use a fixed rounding distribution (namely, standard stochastic quantization). In this section, we outline the theoretical justification for the use of this distribution. We define new objectives for the problem of optimizing the rounding distribution given a fixed quantization set. In particular, we state Theorem 1, which shows that standard stochastic quantization is optimal for minimizing worst-case inner product variance, and Theorem 2, which shows it is NP-hard to find the optimal distribution for minimizing average-case inner product variance.

Given a vector $w \in \mathbb{R}^d$ and quantization set $Q \subseteq \mathbb{R}$, we let Ω denote the set of rounding distributions $\mathcal{D}$ such that $\text{Supp}(\mathcal{D}) \subseteq Q^d$ and $\mathbf{E}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}}[\widehat{\boldsymbol{w}}] = w$. As inner products are linear and $\mathcal{D} \in \Omega$ is unbiased, it is easy to see that for any vector $x \in \mathbb{R}^d$ and $\mathcal{D} \in \Omega$, $\mathbf{E}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}}[\langle \widehat{\boldsymbol{w}}, x \rangle] = \langle w, x \rangle$. We now introduce the new objectives.

Definition 3.1 (WDDV, Worst-Case Distributional Directional Variance)**.** For a vector $w \in \mathbb{R}^d$ and a set $Q \subset \mathbb{R}$, we define

$$\mathsf{WDDV}(w, Q) = \operatorname*{argmin}_{\mathcal{D} \in \Omega} \max_{\substack{x \in \mathbb{R}^d \\ \|x\|_2 \leq 1}} \mathbf{Var}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}}[\langle \widehat{\boldsymbol{w}}, x \rangle].$$

Definition 3.2 (ADDV, Average-Case Distributional Directional Variance)**.** For a vector $w \in \mathbb{R}^d$, a set $Q \subset \mathbb{R}$, and an input distribution $\mathcal{X}$ over $\mathbb{R}^d$, we define

$$\mathsf{ADDV}_{\mathcal{X}}(w, Q) = \operatorname*{argmin}_{\mathcal{D} \in \Omega} \mathbf{E}_{\boldsymbol{x} \sim \mathcal{X}} \left[\mathbf{Var}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}}[\langle \widehat{\boldsymbol{w}}, \boldsymbol{x} \rangle] \right]$$

In Appendix B, we prove Theorem 1 – that standard stochastic quantization is optimal for WDDV. In contrast, in Appendix C, we prove Theorem 2 – that finding the optimal rounding distribution for ADDV is NP-Hard.

Theorem 1. *For every vector $w \in \mathbb{R}^d$ and quantization set $Q \subset \mathbb{R}$, the rounding distribution $\mathcal{D}^{\mathsf{SSQ}}(w, Q)$ minimizes $\mathsf{WDDV}(w, Q)$.*

Theorem 2. *There exists a family of instances, with $w = 0$, $Q = \{-1, 1\}$, and Gaussian input distribution $\mathcal{X}$, on which computing $\mathsf{ADDV}_{\mathcal{X}}(w, Q)$ is NP-Hard.*

The proof of Theorem 1 proceeds by transforming an arbitrary distribution into $\mathcal{D}^{\mathsf{SSQ}}$ and arguing that this transformation cannot have increased the worst-case variance. Theorem 2 follows by a reduction from Maximum Cut on unweighted graphs. These results further motivate the use of standard stochastic quantization as the fixed rounding distribution in MDV and ADV.

3.2 Algorithms for ADV

3.2.1 Exact Algorithms

In the recent literature, [ZLK+17] first defines the standard ASQ problem and presents a simple dynamic programming algorithm with runtime $O(d^2 s)$ and space $O(d^2)$. [BBBIMV24] then present an algorithm named `QUIVER`, which couples the dynamic programming solution with efficient matrix-searching techniques to solve the standard ASQ problem. This improved algorithm has runtime $O(d \log d + ds)$ and uses $O(ds)$ space to output the optimal quantization set.

In a different direction, the survey paper of [GLM+17] outlines many results from the 1-dimensional k-means clustering literature. Perhaps intuitively, as the k-means cost is a lower bound on the MSE of any quantization scheme,[5] k-means has many similarities with minimizing the MSE of an (unbiased) quantization scheme. Indeed, the algorithms mentioned in [GLM+17] can be easily modified to give algorithms for the standard ASQ problem. For example, in the context of 1-dimensional k-means, the dynamic programming and matrix-searching algorithm of [BBBIMV24] (see Corollary D.6) first appeared in [Wu91].

[5] In fact, if one allows biased quantization, the optimal scheme is to compute the k-means clustering and assign each element of w to the nearest cluster.

More generally, [GLM+17] survey a spectrum of theoretical results for k-means that complete the state-of-the-art optimality profile in differing regimes of k and d. Importantly for our work, these algorithms can all be applied to any problem whose structure satisfies the Concave Monge property (Definition A.4), including but not limited to ADV and 1-dimensional k-median clustering in Euclidean space.

A key primitive in these algorithms is finding the shortest path with length (i.e. the number of edges) k in a directed acyclic graph (DAG) whose weights satisfy the Concave Monge property. For this problem, [AST93] gives (i) an algorithm with runtime $O(d\sqrt{k \log d})$, and (ii) an algorithm with runtime $O(d \log \Delta)$ where Δ is the difference between the min and max weight. When $k = \Omega(\log d)$, [Sch98] gives an algorithm with runtime $d2^{O(\sqrt{\log \log d \log k})}$. At a high level, these algorithms work by solving the following *regularized* version of the problem:

$$\min_{\{i_1,\ldots,i_\ell\} \subseteq [d]} \sum_{j=1}^{\ell-1} C[i_j, i_{j+1}] + \tau \ell$$

where $C[i, j]$ is the weight of edge (i, j). That is, we remove the hard constraint of length k edges, and instead add a penalty term to the cost function of τ per edge. By iterating through values of τ appropriately, we can then solve the unregularized (i.e. length-constrained) variant. In solving this regularized problem, [AST93, Sch98] use ideas and results from [Wil88].

These algorithms can be directly applied to ADV since its structure is Concave Monge (see Lemma D.3). As we show in Section 4, an out-of-the-box implementation by [GLM+17] of algorithm (ii) from above outperforms `QUIVER` from [BBBIMV24].

3.2.2 Approximation Algorithms

An additional result of [BBBIMV24] is a bi-criteria approximation algorithm for the *unweighted* MSE objective. In particular, their algorithm returns a quantization set of size $2s - 2$ whose quality is an additive approximation to that of the optimal. It is easy to see, following their analysis, that the same algorithm, analyzed under the *weighted* MSE objective, results in the following weak additive approximation guarantee.

Corollary 3.1 (Lemma 6.1 of [BBBIMV24])**.** *There exists an algorithm such that for any input distribution $\mathcal{X}$ and $m, s \in \mathbb{N}$, it has runtime $O(d + ms)$ and returns a quantization set Q such that $|Q| = 2s-2$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) \leq \mathsf{ADV}_{\mathcal{X}}(w, s) + \sum_{i=1}^{d} \lambda_i \cdot \Delta^2/m^2$. Here, $\Delta := \max_{i,j \in [d]} w_i - w_j$.*

We develop approximation algorithms for optimizing $\mathsf{ADV}_{\mathcal{X}}$ (weighted MSE) with substantially stronger approximation guarantees than that of Corollary 3.1. Furthermore, our algorithms are not bi-criteria. We first give an s-approximation,[6] then improve it to a $(1 + \varepsilon)$-approximation.

Theorem 4. *There exists an algorithm that for any given vector $w \in \mathbb{R}^d$, target quantization set size $s \in \mathbb{N}$, input distribution $\mathcal{X}$, and $\varepsilon > 0$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) \leq (1 + \varepsilon) \cdot \mathsf{ADV}_{\mathcal{X}}(w, s)$. The runtime of the algorithm is $O(d \log(d/\varepsilon))$.*

The first s-approximation algorithm works by exactly solving a suitably defined objective MixDV whose optimal value is an s-approximation to that of ADV. Crucially, the algorithm uses the fact that the objective has a sortedness (Definition A.1) property, which allows for fast k-selection algorithms to be used (see Appendix A). The $(1 + \varepsilon)$-approximation algorithm searches for the Lagrangian multiplier τ of the regularized shortest path on DAG problem described earlier. Importantly, it is *warm-started* using the solution from the s-approximation algorithm, which bounds both the range of this search and the precision to which τ must be found. If no value of τ at this precision yields a path of the desired length, the optimal paths on either side are combined using the Concave Monge property. See Appendix D for details.

Along the way, we observe that the main ideas from solutions to MDV and $\mathsf{ADV}_{\mathcal{X}}$ (in particular, building data-dependent coresets) can be used to improve the bi-criteria approximation algorithm for quantization under the *unweighted* MSE objective. Our algorithm reports a quantization set of size $2s - 2$ with objective value at most $(1 + \varepsilon)$ times optimal in time $O(d \log s + s\sqrt{d/\varepsilon} \log(d/\varepsilon))$; see Theorem 5 in Appendix E for details.

[6] This first approximation algorithm underpins our empirically fast algorithms (which have similar provable approximation guarantees; see Section 4.3).

3.3 Algorithms for MDV

An ideal algorithm for MDV would efficiently return an optimal quantization set. Unfortunately, we prove in Lemma F.22 that the optimal quantization set may contain irrational values (even when the input vector is integer), making such an algorithm impossible under standard finite bit representation (i.e. floating point) of any precision. One must then settle for algorithms which are optimal up to some precision ε.

Theorem 6. *There exists a randomized algorithm that for any given vector $w \in \mathbb{R}^d$, target quantization set size $s \in \mathbb{N}$, and $\varepsilon > 0$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{MDV}(w, Q) \leq (1+\varepsilon) \cdot \mathsf{MDV}(w, s)$. The runtime of the algorithm is $O(d \log(s/\varepsilon))$ with probability 0.99.*

Note that this algorithm does *not* require sorting, and for moderate values of s, ε is faster than the $O(d \log d)$ time required to sort the input vector (as in [BBBIMV24]). Moreover, the dependence on $1/\varepsilon$ is logarithmic, i.e. linear in the number of bits of precision one desires.

This randomized algorithm works by constructing a small *coreset* of values, on which we can solve while still obtaining a close-to-optimal solution for the entire vector. In particular, we show how to construct a subset $y \subseteq w$ of at most $O(s/\sqrt{\varepsilon})$ elements of the input vector such that $\mathsf{MDV}(y, s)$ is within a $(1+\varepsilon)$ factor of $\mathsf{MDV}(w, s)$. Thus, we construct such a subset and then run a slower but still near-linear time algorithm on this coreset.

To construct this coreset, we prove several structural lemmas about MDV and employ faster approximation algorithms for k-center clustering (see Appendix A). To solve on the coreset, we first find a coarse approximation by restricting the quantization points to be values from the input vector, and then use this estimate to "warm-start" a binary search for the optimal maximal variance (again, over the coreset). The most technically difficult step is finding this coarse approximation: it bears some similarities to the fast matrix search algorithms employed to optimize for ADV, but there are a number of added difficulties specific to the max objective of MDV. See Appendix F for the full details of the algorithm and the proof of Theorem 6.

4 Empirical Results

4.1 Empirical Setup

We implement a number of algorithms in optimized `Cython` code[7]. We also modify and extend the code of [GLM+17] in our exact algorithms for ADV/weighted ASQ. Our experiments were run on a Mac Mini consumer desktop computer with an Apple M4 processor and 16GB of RAM. In the supplementary information, we include empirical results from 33 different data distributions; each is qualitatively similar to the results presented in this section.

4.2 Faster Exact Algorithms for ASQ

Our first contribution is significantly faster algorithms for ASQ[8] which, in some regimes, obtain a $10\times$ speedup over the previous state-of-the-art accelerated `QUIVER` from [BBBIMV24]. Our implementations can be used as a drop-in replacement for `QUIVER`, and thus speed up any algorithm using `QUIVER`; in particular, we give faster approximation algorithms (see Section D.3).

Our first insight to make these speedups is that the fast exact 1D k-means algorithms presented in [GLM+17] can be modified to solve ASQ; see Section D.1 for the full details. This strategy already gives algorithms which are faster than `QUIVER`; namely, the `Wilber` algorithm from [GLM+17], when run with interpolation search and modified to support the ASQ objective, is faster than `QUIVER` in many regimes (see Figure 3).

To improve the performance further, we exploit the inner workings of `Wilber` to *warm-start* the algorithm with an approximation. This allows the binary search of `Wilber` to converge much faster, leading to additional speedups. In particular, `Wilber` works by solving a shortest path on a directed acyclic graph (DAG), whose weights are determined in part by a Lagrangian multiplier τ.

[7] Our code is availiable here: `https://github.com/nathanllww/Inner-Product-Aware-Quantization`

[8] Our algorithms, like `QUIVER`, support weights, as needed to solve ADV.

Unfortunately, [GLM+17] show that the desired value τ is the difference between the optimal ASQ cost with s quantization points and the cost with $s+1$ quantization points, so it must be found via binary search.

To speed up this search, we use a fast yet robust approximation algorithm (`MixApprox`, see Section D.3) to estimate the value τ, and only search in the range around this estimate. Letting τ' be the difference in costs returned by `MixApprox` when run with s and $s+1$ quantization points, we only search for $\tau \in [\tau'/2, 2\tau']$ (reverting to searching all possible values if nothing is found). Since `MixApprox` is an accurate approximation algorithm across a range of distributions, τ is found very quickly, leading to a large speedup. We plot the runtime of our method against `QUIVER` and vanilla `Wilber` in Figure 3.

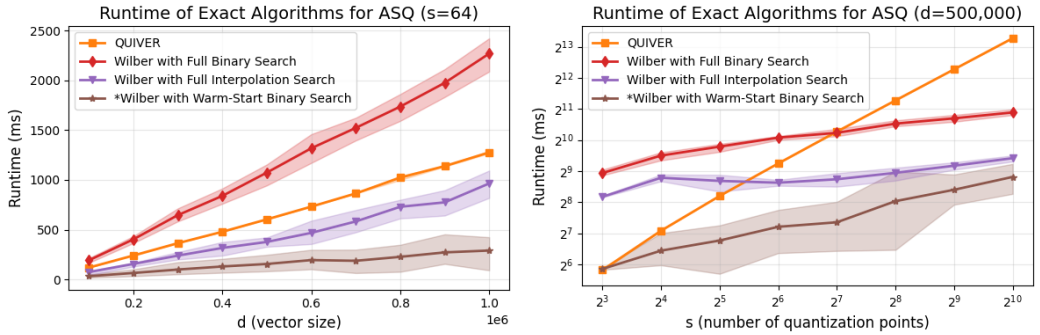


Figure 3: Runtime of exact algorithms for ASQ on vectors w of dimension d drawn from LogNormal$(0,1)$. The shaded regions show the 10–90% range of runtimes across draws of w. Orange denotes accelerated `QUIVER`; red denotes `Wilber` with binary search over the full range of possible τ; purple denotes `Wilber` with the interpolation search method of [GLM+17]; and brown denotes our accelerated search method.

4.3 Faster Approximation Algorithms for ASQ

The approximate `QUIVER` algorithm of [BBBIMV24] operates by first placing m evenly spaced markers between the minimum and maximum entries of the vector w, and then finds the optimal s of these m discretized points to include as quantization values. Unsurprisingly, then, using our faster exact algorithms to find the optimal s quantization values, we are able to improve the performance[9] of the approximate `QUIVER` algorithm as well; see Figure 4.

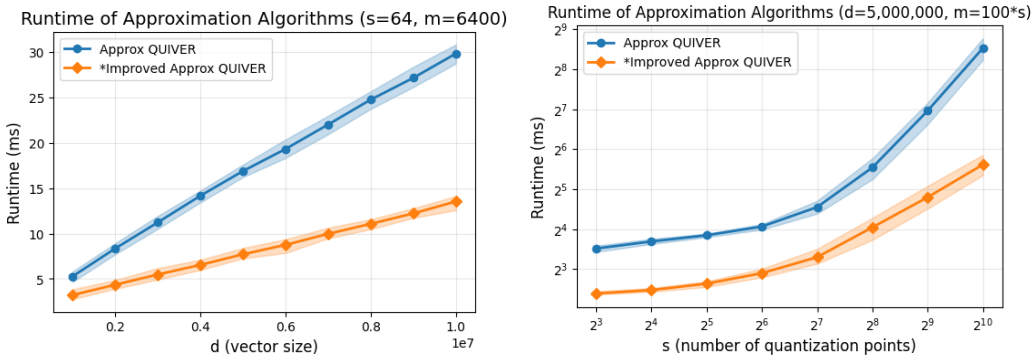


Figure 4: Runtime comparison of approximate `QUIVER` and our implementation using faster exact subroutines on vectors w drawn from LogNormal$(0,1)$, with $m = 100s$ uniformly spaced discretization points. The shaded regions show the 10–90% range of runtimes across draws of w.

[9] Due to the special structure of the calls approximate `QUIVER` makes to the exact algorithm, we use a slight variant of our fast exact algorithm; see Section D.3 for details. The overall structure of the algorithm is the same.

4.4 Algorithms for MDV

We also give a fast practical algorithm for MDV, inspired by our theoretical algorithms; see Section F.8 for details. In the first step when run on vector w, the algorithm constructs a coreset in a manner similar to approximate QUIVER: it divides the range $[\min_i w_i, \max_i w_i]$ into $10s$ equally sized buckets, and constructs the coreset by taking the minimum and maximum value of w from each bucket. The algorithm then uses binary search to find the worst-case variance, by "guessing and checking". To do the check, we utilize an algorithm which (quickly) computes the minimum quantization set size required to obtain a target worst-case variance of v.

However, if any bucket is larger than $\sqrt{\varepsilon v_{\max}}$, where $v_{\max}$ is the current maximum of the search range, the algorithm *subdivides* the buckets into smaller sizes until all have length at most $\sqrt{\varepsilon v_{\max}}$. This subdivision step is the crucial modification that allows for a guaranteed $(1+\varepsilon)$-approximation, even on heavily skewed data, but the adaptive nature also keeps the algorithm very fast. In a certain sense, the algorithm is adaptively finding the *instance ideal* bucket size, keeping it as fast as possible for a given desired accuracy. The proof of the $(1+\varepsilon)$-approximation can be found in Lemma F.23.

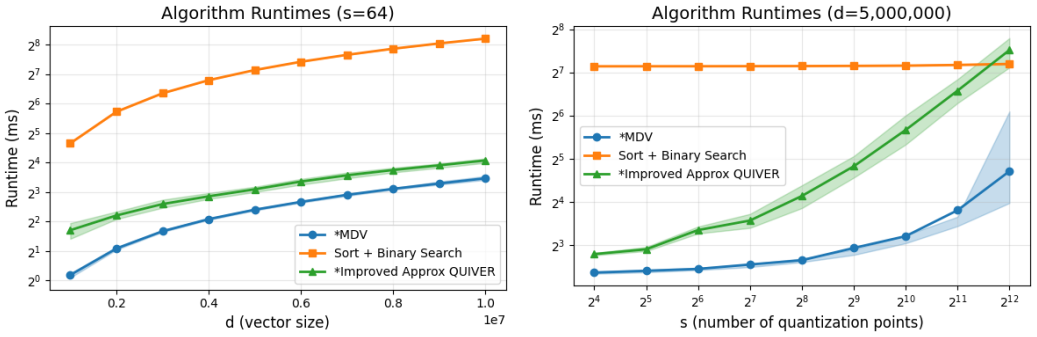


Figure 5: Runtime of algorithms for MDV compared against Improved Approx QUIVER as a baseline (although it does not provide a good approximation to MDV), on vectors drawn from LogNormal$(0,1)$. The shaded regions show the 10–90% range of runtimes across draws of the input vector.

In Figure 5, the algorithm labeled "MDV" is this fast algorithm, run with $\varepsilon = 0.01$, while the "Sort + Binary Search" method sorts and then uses binary search to find the smallest v for which s quantization points can achieve worst-case variance v. Improved Approx QUIVER is always run with quality parameter $m = 100s$. Note that this implementation of Improved Approx QUIVER, in contrast to the implementation of Approx QUIVER provided by [BBBIMV24] and our implementation shown in Figure 4, does not require the input vector to be sorted.

4.5 LLM Quantization

We evaluate our quantization schemes against several popular options on the task of LLM quantization. In particular, we quantize the MLP weights of the open-weight model Qwen3.5-4B [Qwe26]; the full details of the method used for quantizing can be found in Appendix G.

Given input vector x_{in}, each MLP layer computes

$$x_{\text{out}} = W_{\text{down}}\left(f_{\text{act}}(W_{\text{gate}}x_{\text{in}}) \otimes W_{\text{up}}x_{\text{in}}\right)$$

where $W_{\text{down}}, W_{\text{gate}}, W_{\text{up}}$ are the (learned) weight vectors, $\otimes$ denotes element-wise multiplication, and f_{act} is a nonlinear activation function applied element-wise (in this case, SiLU [Qwe26]).

Note each weight matrix participates in only matrix-vector products, which should be thought of as inner products of an arbitrary query vector with each row of the weight matrix. It is natural to apply our vector quantization scheme to each row individually; see Table 1.

However, in practice one may prefer to construct one quantization set per weight matrix to lower the codebook construction time and lower the average bits per parameter. We also test this alternative method. Despite the fact that our methods are designed with vector quantization in mind, we find that they yield strong results in the matrix quantization setting as well (see Table 2).

Table 1: Per-row quantization of the MLP weights of Qwen3.5-4B with $s = 32$. KL divergence, top-1 agreement, and perplexity are measured on WikiText-2, and KL and top-1 agreement are computed relative to the unquantized model. ARC Acc. is accuracy on the ARC-Challenge benchmark of basic science questions, and ARC agree is agreement with the unquantized model's answers. ADV and ASQ round to nearest rather than stochastically; TurboQuant is from [ZDHM25]. $*$ denotes our method.

Method	**KL** ($\times 10^{-2}$)	**Top-1 (%)**	**PPL**	**ARC Acc. (%)**	**ARC Agree (%)**
Unquantized	0	100	9.58	54.86	100
ADV*	**1.17**	**95.46**	9.63	54.44	**96.33**
ASQ	1.36	95.11	**9.58**	54.69	96.08
k-means	1.37	95.10	9.65	53.92	95.48
TurboQuant	1.37	95.00	9.62	**55.38**	95.31
Uniform	2.89	92.56	9.75	54.10	94.37

Table 2: The same evaluation as Table 1, with one quantization set per weight matrix rather than per row.

Method	**KL** ($\times 10^{-2}$)	**Top-1 (%)**	**PPL**	**ARC Acc. (%)**	**ARC Agree (%)**
Unquantized	0	100	9.58	54.86	100
ADV*	**2.29**	**93.51**	9.63	55.55	93.86
ASQ	2.50	93.25	9.74	55.63	**95.14**
k-means	6.37	89.86	9.78	54.69	94.88
TurboQuant	6.06	90.33	**9.25**	**56.40**	93.60
Uniform	185.32	43.64	58.62	33.36	54.35

5 Discussion

Limitations. We now discuss a few important limitations of our work.

- Although MDV outperforms ADV on worst-case variance, our experimentation revealed that it is outperformed by ADV on the worst 1% of tail events for many distributions. This perhaps motivates future work studying a *hybrid* objective that optimizes for the worst 1% performance over a given input distribution $\mathcal{X}$.
- Our largest speed improvements over the prior state-of-the-art `QUIVER` algorithm come when the number of quantization points s is large. It is natural to consider whether we can adapt our algorithms to achieve similar speedup over `QUIVER` for small values of s as well.

Future Work. We conclude with some interesting open questions that are natural extensions of this work:

- Theorem 2 shows that computing the optimal rounding distribution for the average-case problem is NP-Hard. Do there exist efficient algorithms to find approximately optimal rounding distributions?
- Applying this notion of adaptive stochastic quantization to vector search requires the storage of a quantization set for every vector w in the dataset. Can our ideas and algorithms be adapted to (potentially dynamic) *multi-vector* quantization settings?

Acknowledgments

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-23-F-0014 in the amount of $140,600. KS is supported by the National Science Foundation (NSF) under Grant No. CCF-2337993. The authors thank the anonymous reviewers whose suggestions improved the quality and presentation of this work.

References

[AGL+17] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[AKM+86] Alok Aggarwal, Maria Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix searching algorithm. In *Proceedings of the second annual symposium on Computational geometry*, pages 285–292, 1986.

[AST93] Alok Aggarwal, Baruch Schieber, and Takashi Tokuyama. Finding a minimum weight k-link path in graphs with monge property and applications. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 189–197, 1993.

[BBBIMV24] Ran Ben-Basat, Yaniv Ben-Itzhak, Michael Mitzenmacher, and Shay Vargaftik. Optimal and approximate adaptive stochastic quantization. *Advances in Neural Information Processing Systems*, 37:94265–94291, 2024.

[BBBIMV25] Ran Ben Basat, Yaniv Ben-Itzhak, Michael Mitzenmacher, and Shay Vargaftik. Better than optimal: Improving adaptive stochastic quantization using shared randomness. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 9(3):1–44, 2025.

[BNS19] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in neural information processing systems*, 32, 2019.

[CBM25] Riley Carlson, John Bauer, and Christopher D. Manning. A new pair of gloves, 2025.

[CBUS+20] Brian Chmiel, Liad Ben-Uri, Moran Shkolnik, Elad Hoffer, Ron Banner, and Daniel Soudry. Neural gradients are near-lognormal: improved quantized and sparse training. *arXiv preprint arXiv:2006.08173*, 2020.

[FAHA23] E Frantar, S Ashkboos, T Hoefler, and D Alistarh. Optq: Accurate quantization for generative pre-trained transformers. 2023. In *URL https://openreview. net/forum*, 2023.

[FG88] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 434–444, New York, NY, USA, 1988. Association for Computing Machinery.

[FHH+20] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *International Conference on Machine Learning*, pages 3304–3314. PMLR, 2020.

[FJ84] Greg N Frederickson and Donald B Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal on computing*, 13(1):14–30, 1984.

[FTM+20] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *Advances in neural information processing systems*, 33:3174–3185, 2020.

[GGX+25] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 3(3):1–26, 2025.

[GHKS13] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2013.

[GJS74] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.

[GLM+17] Allan Grønlund, Kasper Green Larsen, Alexander Mathiasen, Jesper Sindahl Nielsen, Stefan Schneider, and Mingzhou Song. Fast exact k-means, k-medians and bregman divergence clustering in 1d. *arXiv preprint arXiv:1701.07204*, 2017.

[Gon85] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[HJ12] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2012.

[JLPK23] Yongkweon Jeon, Chungman Lee, Kyungphil Park, and Ho-young Kim. A frustratingly easy post-training quantization scheme for llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14446–14461, 2023.

[MA85] Andranik Mirzaian and Eshrat Arjomandi. Selection in x+ y and matrices with sorted rows and columns. *Information processing letters*, 20(1):13–17, 1985.

[MNA+17] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[MUJS18] Yusuke Matsui, Yusuke Uchida, Herve Jegou, and Shin'ichi Satoh. A survey of product quantization. *ITE Transactions on Media Technology and Applications*, 6(1):2–10, 2018.

[NAVB+20] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International conference on machine learning*, pages 7197–7206. PMLR, 2020.

[Qwe26] Qwen Team. Qwen3.5: Towards native multimodal agents. Available as `https://qwen.ai/blog?id=qwen3.5`, February 2026.

[RKFM+21] Ali Ramezani-Kebrya, Fartash Faghri, Ilya Markov, Vitalii Aksenov, Dan Alistarh, and Daniel M Roy. Nuqsgd: Provably communication-efficient data-parallel sgd via nonuniform quantization. *Journal of Machine Learning Research*, 22(114):1–43, 2021.

[Sch98] Baruch Schieber. Computing a minimum weightk-link path in graphs with the concave monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

[SZY+23] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[VAM18] Mariia Vladimirova, Julyan Arbel, and Pablo Mesejo. Bayesian neural networks become heavier-tailed with depth. In *NeurIPS 2018-Thirty-second Conference on Neural Information Processing Systems*, pages 1–7, 2018.

[Wil88] Robert Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.

[Wu91] Xiaolin Wu. Optimal quantization by matrix searching. *Journal of algorithms*, 12(4):663–673, 1991.

[ZDHM25] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. *arXiv preprint arXiv:2504.19874*, 2025.

[ZLK+17] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning*, pages 4035–4043. PMLR, 2017.

Appendix Contents

A Technical Preliminaries **13**

B Standard Stochastic Quantization is Optimal for WDDV **14**

C NP-Hardness of Optimizing ADDV **16**

D Average Directional Variance **18**

E An Improved Approximation Algorithm for Unweighted MSE **26**

F Maximum Directional Variance **29**

G LLM Quantization Setup **41**

H Deferred Proofs and Figures **42**

A Technical Preliminaries

A.1 Matrix Properties and Algorithms

Definition A.1 (Sorted Matrix)**.** We call a matrix $A \in \mathbb{R}^{d\times d}$ *sorted* if all its rows are sorted in non-decreasing order while all columns are in non-increasing order.

Lemma A.1 (`Sorted-Selection` Algorithm, Theorem 1 of [FJ84] and Corollary 6.2 of [MA85])**.** *Let $A \in \mathbb{R}^{d\times d}$ be a sorted matrix. Then, there exists an algorithm that returns the kth smallest element (i.e., performs k-selection) in $O(d)$ time.*

Importantly, the algorithms of Lemma A.1 operate in the query access model, where they are allowed $O(1)$ time access to any entry of A. The algorithms from [FJ84, MA85] achieve $O(d)$ time for k-selection in row-and-column sorted $d \times d$ matrices by utilizing a divide-and-conquer strategy that systematically restricts the search space to a narrow "staircase" contour of $O(d)$ candidate elements. [FJ84] accomplishes this through recursive matrix bisection into 2×2 submatrices, extracting representatives to compute bounds that strictly contain the k-th element, and then applying a linear-time selection subroutine to the remaining candidates along the monotonic frontier. [MA85] streamlines this recursive procedure and greatly simplifies the algorithm and analysis.

Definition A.2 (Monotone Matrix)**.** Let matrix $A \in \mathbb{R}^{d\times d}$ and define $c(i) := \min\{j \in [d] : A[i,j] = \min_{k\in[d]} A[i,k]\}$. Matrix A is monotone if for all $i \in [d]$, $c(i) \le c(i+1)$.

Definition A.3 (Totally Monotone Matrix)**.** Matrix $A \in \mathbb{R}^{d\times d}$ is *totally* monotone if every submatrix of A is monotone. Equivalently, A is totally monotone if for $i < i'$ and $j < j'$, $A[i,j] > A[i,j'] \implies A[i',j] > A[i',j']$.

Definition A.4 (Concave Monge Property)**.** Matrix $A \in \mathbb{R}^{d\times d}$ satisfies the concave Monge property if for any $1 \le j < k < d$, $A[j,k] + A[j+1,k+1] \le A[j,k+1] + A[j+1,k]$.

Definition A.5 (Quadrangle Inequality)**.** Matrix $A \in \mathbb{R}^{d\times d}$ satisfies the quadrangle inequality if for any $a \le b \le c \le f$, $A[a,c] + A[b,f] \le A[a,f] + A[b,c]$.

It is clear that any matrix satisfying the Quadrangle Inequality (i) satisfies the Concave Monge property, (ii) is totally monotone, and (iii) is monotone.

Lemma A.2 (SMAWK Algorithm, Theorem 4.3 of [AKM+86]). *Let $A \in \mathbb{R}^{d \times d}$ be a totally monotone matrix. Then, there exists an algorithm that in $O(d)$ time returns $j(i) := \min\{j \in [d] : A[i,j] = \max_{k \in [d]} A[i,k]\}$ for all rows $i \in [d]$.*

The SMAWK algorithm leverages the total monotonicity property to systematically eliminate redundant columns, thereby shrinking the matrix width. Following this, the algorithm recurses on the even-indexed rows of the reduced matrix to find their respective minima. In the final step, it uses the computed minima of the even rows to tightly constrain the search space for the odd-indexed rows, locating the remaining minima in strictly linear time.

A.2 Clustering Algorithms

Many algorithms in Appendices E and F make use of exact and/or approximate clustering algorithms as subroutines. These clustering algorithms are often used to give *data-dependent coresets* of the original instance, upon which further processing is done.

In particular, we utilize an algorithm for s-center clustering in Euclidean space. Here, an input consists of d vectors $v_1, \ldots, v_d \in \mathbb{R}^n$ and the task is to return an s-clustering $\mathcal{C} = (C_1, \ldots, C_s)$ such that $C_1 \sqcup \ldots \sqcup C_s = \{v_1, \ldots, v_d\}$ that minimizes the maximum radius of any cluster.

$$\text{Radius}(\mathcal{C}) := \frac{1}{2} \cdot \max_{i \in [s]} \max_{v_j, v_k \in C_i} \left\| v_j - v_k \right\|_2$$

The following 2-approximate s-center clustering algorithm of [FG88] is used often.

Lemma A.3 (`s-Center-Clustering` Approximation Algorithm, Theorem 4.1 of [FG88]). *Given d vectors $v_1, \ldots, v_d \in \mathbb{R}^n$ and a target number of clusters $s \in \mathbb{N}$, there exists an algorithm with runtime $O(d \log s)$ that returns an s-clustering $\mathcal{C}$ such that*

$$\text{Radius}(\mathcal{C}) \leq 2 \cdot \min_{\substack{\mathcal{A} = (A_1, \ldots, A_s) \\ A_1 \sqcup \ldots \sqcup A_s = \{v_1, \ldots, v_d\}}} \text{Radius}(\mathcal{A})$$

The `s-Center-Clustering` algorithm of Lemma A.3 works similarly to Gonzalez's algorithm [Gon85] by finding an independent set in a suitable implicitly defined geometric graph. This graph, however, is defined over *boxes* of points rather than the points themselves. This ultimately allows for a more efficient data structure that can be used to construct the independent set. The work is technical and involved, so for further details, consult the original paper [FG88].

Hidden in the lemma statement is that the runtime of the above algorithm has exponential dependence on the dimension of the space n. In this work, however, we only utilize this algorithm on 1-dimensional instances.

Remark A.1. The result of Lemma A.3 is largely theoretical due to large constant factors and overall complexity of the data structures involved in the algorithm. As such, any algorithm using it as a subroutine is most-probably impractical to implement. However, empirically implemented algorithms are heavily inspired by the theoretical results and are outlined in each relevant appendix.

B Standard Stochastic Quantization is Optimal for WDDV

In this section, we prove that the standard stochastic quantization rounding distribution is optimal for WDDV (Definition 3.1). Recall that $\mathcal{D}^{\mathsf{SSQ}}(w, Q)$ is the distribution that rounds each point w_i to either $w_i^{\uparrow}(Q)$ or $w_i^{\downarrow}(Q)$ independently, with probabilities chosen to ensure that the rounding is unbiased.

Theorem 1. *For every vector $w \in \mathbb{R}^d$ and quantization set $Q \subset \mathbb{R}$, the rounding distribution $\mathcal{D}^{\mathsf{SSQ}}(w, Q)$ minimizes $\mathsf{WDDV}(w, Q)$.*

To prove Theorem 1, we first show the following key lemma.

Lemma B.1. *Consider $u \in \mathbb{R}$ and quantization set $Q \subseteq \mathbb{R}$. Let $u^{\downarrow} := \max\{q \in Q : q \leq u\}$, $u^{\uparrow} := \min\{q \in Q : q \geq u\}$, and $\mathcal{D}^*$ denote the unique distribution that rounds u to $\{u^{\downarrow}, u^{\uparrow}\}$ such that $\mathbf{E}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^*}[\widehat{\boldsymbol{u}}] = u$. Then, for any distribution $\mathcal{D}$ such that $\text{supp}(\mathcal{D}) \subseteq Q$ and $\mathbb{E}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}] = u$, we have that $\mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}] \geq \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^*}[\widehat{\boldsymbol{u}}]$.*

Proof. Let $Q^{\downarrow} := \text{supp}(\mathcal{D}) \cap \{q \in Q : q \leq u\}$ and analogously $Q^{\uparrow} := \text{supp}(\mathcal{D}) \cap \{q \in Q : q \geq u\}$. The argument proceeds by first constructing an unbiased distribution $\mathcal{D}'$ supported only on the centers of mass of $Q^{\downarrow}$ and $Q^{\uparrow}$ and showing that $\mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] \leq \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}]$. We then show that $\mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^*}[\widehat{\boldsymbol{u}}] \leq \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}]$ to conclude the claim. First, define the centers of mass

$$Q^{\downarrow}_{\text{COM}} := \sum_{q \in Q^{\downarrow}} q \cdot \frac{\mathcal{D}(q)}{\mathcal{D}(Q^{\downarrow})}, \qquad Q^{\uparrow}_{\text{COM}} := \sum_{q \in Q^{\uparrow}} q \cdot \frac{\mathcal{D}(q)}{\mathcal{D}(Q^{\uparrow})}$$

where $\mathcal{D}(S) \triangleq \sum_{v \in S} \mathcal{D}(v)$ for $S \subseteq Q$. Define distribution $\mathcal{D}'$ by placing mass $\mathcal{D}(Q^{\downarrow})$ at $Q^{\downarrow}_{\text{COM}}$ and mass $\mathcal{D}(Q^{\uparrow})$ at $Q^{\uparrow}_{\text{COM}}$ as follows[10]

$$\mathcal{D}'(x) = \begin{cases} \mathcal{D}(Q^{\downarrow}) \text{ if } x = Q^{\downarrow}_{\text{COM}} \\ \mathcal{D}(Q^{\uparrow}) \text{ if } x = Q^{\uparrow}_{\text{COM}} \\ 0 \text{ otherwise} \end{cases}$$

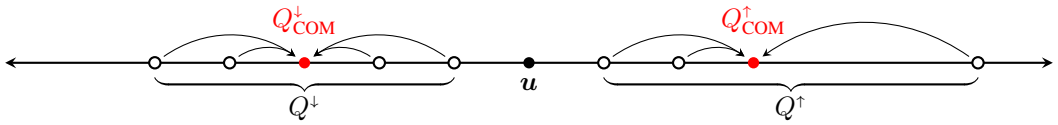


Figure 6: Illustration of distribution $\mathcal{D}'$, showing the reduction of support of $\mathcal{D}$ to just two elements

See Figure 6 for an illustration of the transformation from distribution $\mathcal{D}$ to $\mathcal{D}'$. Notice that

$$\begin{aligned} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] = Q^{\downarrow}_{\text{COM}} \cdot \mathcal{D}(Q^{\downarrow}) + Q^{\uparrow}_{\text{COM}} \cdot \mathcal{D}(Q^{\uparrow}) &= \sum_{q \in Q^{\downarrow}} q \cdot \mathcal{D}(q) + \sum_{q \in Q^{\uparrow}} q \cdot \mathcal{D}(q) \\ &= \sum_{q \in Q} q \cdot \mathcal{D}(q) = \mathop{\mathbf{E}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}] = u \end{aligned}$$

Bounding the variance,

$$\begin{aligned} \mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] &= \mathop{\mathbf{E}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}^2] - \mathop{\mathbf{E}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}]^2 \\ &= [\mathcal{D}(Q^{\downarrow}) \cdot {Q^{\downarrow}_{\text{COM}}}^2 + \mathcal{D}(Q^{\uparrow}) \cdot {Q^{\uparrow}_{\text{COM}}}^2] - u^2 \\ &= \left[\frac{\left(\sum_{q \in Q^{\downarrow}} q \cdot \mathcal{D}(q)\right)^2}{\mathcal{D}(Q^{\downarrow})} + \frac{\left(\sum_{q \in Q^{\uparrow}} q \cdot \mathcal{D}(q)\right)^2}{\mathcal{D}(Q^{\uparrow})} \right] - u^2 \\ &\leq \left[\sum_{q \in Q^{\downarrow}} q^2 \mathcal{D}(q) + \sum_{q \in Q^{\uparrow}} q^2 \mathcal{D}(q) \right] - u^2 \qquad \text{(Cauchy-Schwarz Inequality)} \\ &= \left[\sum_{q \in Q} q^2 \mathcal{D}(q) \right] - u^2 = \mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}] \end{aligned}$$

and thus $\mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] \leq \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}]$. Because $\mathbf{E}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] = u$, it follows that $\mathcal{D}'$ performs unbiased rounding of u to the set $\{Q^{\downarrow}_{\text{COM}}, Q^{\uparrow}_{\text{COM}}\}$. Moreover, as $Q^{\downarrow}_{\text{COM}} \leq u^{\downarrow}$ and $Q^{\uparrow}_{\text{COM}} \geq u^{\uparrow}$ by definition of the center of mass, we have

$$\mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] = (u - Q^{\downarrow}_{\text{COM}})(Q^{\uparrow}_{\text{COM}} - u) \geq (u - u^{\downarrow})(u^{\uparrow} - u) = \mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^*}[\widehat{\boldsymbol{u}}]$$

and so

$$\mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^*}[\widehat{\boldsymbol{u}}] \leq \mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}'}[\widehat{\boldsymbol{u}}] \leq \mathop{\mathbf{Var}}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}]. \qquad \square$$

[10] Note that because $\mathcal{D}'$ is an intermediate distribution only used for the purposes of the analysis, the fact that $\{Q^{\downarrow}_{\text{COM}}, Q^{\uparrow}_{\text{COM}}\}$ may *not* be a subset of Q does not affect correctness

We are now ready to prove Theorem 1. Here, for a given $w \in \mathbb{R}^d$ and $Q \subseteq \mathbb{R}$, we use the notation $\mathsf{WDDV}(w, \mathcal{D}) := \max_{x:\|x\|_2 \le 1} \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\langle \widehat{\boldsymbol{u}}, x \rangle]$ to denote the cost of choosing rounding distribution $\mathcal{D}$.

Proof of Theorem 1. Observe that

$$\begin{aligned}
\mathsf{WDDV}(w, \mathcal{D}^{\mathsf{SSQ}}(w, Q)) &= \max_{x:\|x\|_2 \le 1} \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\langle \widehat{\boldsymbol{u}}, x \rangle] \\
&= \max_{x:\|x\|_2 \le 1} \sum_{i=1}^{d} x_i^2 \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\widehat{\boldsymbol{u}}_i] \qquad (\mathcal{D}^{\mathsf{SSQ}} \text{ is a product distribution}) \\
&= \max_{i \in [d]} \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\widehat{\boldsymbol{u}}_i] = \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\widehat{\boldsymbol{u}}_{i^*}]
\end{aligned}$$

where $i^* := \operatorname{argmax}_{i \in [d]} \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\widehat{\boldsymbol{u}}_i]$. Define $\Sigma := \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\widehat{\boldsymbol{u}}_{i^*}]$. Now, consider any arbitrary rounding $\mathcal{D} \in \Omega$. It follows that

$$\begin{aligned}
\mathsf{WDDV}(w, \mathcal{D}) \ge \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\langle \widehat{\boldsymbol{u}}, e_{i^*} \rangle] = \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}}[\widehat{\boldsymbol{u}}_{i^*}] &\ge \mathbf{Var}_{\widehat{\boldsymbol{u}} \sim \mathcal{D}^{\mathsf{SSQ}}}[\widehat{\boldsymbol{u}}_{i^*}] \qquad \text{(Lemma B.1)} \\
&= \mathsf{WDDV}(w, \mathcal{D}^{\mathsf{SSQ}}(w, Q))
\end{aligned}$$

as desired. □

Remark B.1. Theorem 1 shows that $\mathcal{D}^{\mathsf{SSQ}}$ is a minimizer of $\mathsf{WDDV}(w, Q)$. Importantly, however, it may not be the *unique* minimizer. It is feasible that there exist non-adaptive and/or dependent rounding schemes that simultaneously minimize $\mathsf{WDDV}(w, Q)$.

C NP-Hardness of Optimizing ADDV

Lemma C.1 (Gershgorin Circle Theorem, Theorem 6.1.1 in [HJ12])**.** *Let $A \in \mathbb{C}^{d \times d}$ and let $R_i(A) := \sum_{j \ne i} |A_{ij}|$ for all $i \in [d]$. Then, the eigenvalues of A are contained in the union of discs*

$$\bigcup_{i \in [d]} \{z \in \mathbb{C} : |z - A_{ii}| \le R_i(A)\}$$

Corollary C.2 (Lower Bound on Minimum Eigenvalue of Real Symmetric Matrices)**.** *Let $A \in \mathbb{R}^{d \times d}$ be a symmetric matrix with eigenvalues $\lambda_1 \ge \ldots \ge \lambda_d$. Then,*

$$\lambda_d \ge \min_{i \in [d]} \left(A_{ii} - \sum_{j \ne i} |A_{ij}| \right)$$

Proof. By Lemma C.1, every eigenvalue of A lies inside the union of discs

$$\bigcup_{i \in [d]} \{z \in \mathbb{C} : A_{ii} - R_i(A) \le z \le A_{ii} + R_i(A)\}$$

It follows that

$$\lambda_d \ge \min_{i \in [d]} \big(A_{ii} - R_i(A) \big) = \min_{i \in [d]} \left(A_{ii} - \sum_{j \ne i} |A_{ij}| \right)$$

□

Lemma C.3. *Let $M \in \mathbb{R}^{d \times d}$ be a symmetric positive semi-definite matrix. Then, there exists a random variable $\boldsymbol{x}$ such that*

$$\mathbf{Cov}(\boldsymbol{x}) = \mathbf{E}[\boldsymbol{x}\boldsymbol{x}^\intercal] = M.$$

Proof. Define $\boldsymbol{z} \sim \mathcal{N}(0, I)$ and $\boldsymbol{x} := M^{1/2}\boldsymbol{z}$. $\boldsymbol{x}$ is well-defined as M is symmetric and positive semi-definite, and thus has a real square root $M^{1/2}$. Therefore,

$$\begin{aligned}
\mathbf{Cov}(\boldsymbol{x}) &= \mathbf{E}[(\boldsymbol{x} - \mathbf{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbf{E}[\boldsymbol{x}])^\intercal] \\
&= \mathbf{E}[\boldsymbol{x}\boldsymbol{x}^\intercal] && (\mathbf{E}[\boldsymbol{x}] = \mathbf{E}[M^{1/2}\boldsymbol{z}] = 0) \\
&= M^{1/2}\,\mathbf{E}[\boldsymbol{z}\boldsymbol{z}^\intercal]M^{1/2} \\
&= M^{1/2}IM^{1/2} = M. && \square
\end{aligned}$$

Theorem 2. *There exists a family of instances, with $w = 0$, $Q = \{-1, 1\}$, and Gaussian input distribution $\mathcal{X}$, on which computing $\mathsf{ADDV}_{\mathcal{X}}(w, Q)$ is NP-Hard.*

Proof. We show a reduction from the decision version of unweighted Max-Cut. Consider an instance given by an unweighted (simple) graph G. The algorithm is as follows.

Algorithm 1: Max-Cut via ADDV

Input: Graph G, Oracle access to $\mathsf{ADDV}_{\mathcal{X}}(w, Q)$
Output: Max-Cut(G)

1 Construct adjacency matrix A of G
2 Let $y := \min_{i\in[d]} \left(A_{ii} - \sum_{j\neq i} |A_{ij}|\right)$ and $M := A - yI$
3 Let $\mathcal{D}^{\text{opt}} := \mathsf{ADDV}_{\mathcal{N}(0,M)}(0, \{-1, 1\})$
4 Let $\widehat{w}^{\text{opt}}$ be an arbitrary vector from the support of $\mathcal{D}^{\text{opt}}$
5 Return $\frac{1}{4}\sum_{i\neq j} M_{ij} - \frac{1}{4}\sum_{i\neq j} \widehat{w}_i^{\text{opt}}\widehat{w}_j^{\text{opt}} M_{ij}$

The adjacency matrix $A \in \mathbb{R}^{d\times d}$ of G is

$$A_{ij} := \begin{cases} 1 \text{ if } (i,j) \in G \\ 0 \text{ otherwise} \end{cases}$$

Then, define $y := \min_{i\in[d]} \left(A_{ii} - \sum_{j\neq i} |A_{ij}|\right)$ and let $M := A - yI$. Observe that M is positive semi-definite and symmetric. To see why this is the case, let (v, λ) be an eigenvector-eigenvalue pair of A. Then,

$$Mv = (A - yI)v = Av - yIv = \lambda v - yv = (\lambda - y)v,$$

so the eigenvalues of M are $\lambda_1 - y \geq \ldots \geq \lambda_d - y$. By Corollary C.2, $\lambda_d \geq y$, and all eigenvalues of M are real and non-negative; hence M is positive semi-definite. Symmetry of M follows immediately from symmetry of A.

We now construct a corresponding instance of ADDV and show that its optimum yields a solution to Max-Cut on G. In particular, consider the instance given by $w = 0 \in \mathbb{R}^d$, $Q = \{-1, 1\}$, and $\mathcal{X} = \mathcal{N}(0, M)$. Note that $\mathcal{X}$ is a valid distribution due to Lemma C.3, and recall that Ω denotes the set of unbiased rounding distributions with support $\subseteq Q^d$. Then, the ADDV objective cost is given by

$$\begin{aligned}
\min_{\mathcal{D}\in\Omega} \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{N}(0,M)} \left[\mathop{\mathbf{Var}}_{\widehat{\boldsymbol{w}}\sim\mathcal{D}}[\langle\widehat{\boldsymbol{w}}, \boldsymbol{x}\rangle]\right] &= \min_{\mathcal{D}\in\Omega} \mathop{\mathbf{E}}_{\substack{\boldsymbol{x}\sim\mathcal{N}(0,M) \\ \widehat{\boldsymbol{w}}\sim\mathcal{D}}} [\langle\widehat{\boldsymbol{w}}, \boldsymbol{x}\rangle^2] && (\mathbf{E}_{\widehat{\boldsymbol{w}}\sim\mathcal{D}}[\langle\widehat{\boldsymbol{w}}, \boldsymbol{x}\rangle] = \langle w, x\rangle = 0) \\
&= \min_{\mathcal{D}\in\Omega} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}}\sim\mathcal{D}} \left[\sum_{i,j=1}^{d} \widehat{\boldsymbol{w}}_i\widehat{\boldsymbol{w}}_j \cdot \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{N}(0,M)} [\boldsymbol{x}_i\boldsymbol{x}_j]\right] \\
&= \min_{\mathcal{D}\in\Omega} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}}\sim\mathcal{D}} \left[\sum_{i=1}^{d} \widehat{\boldsymbol{w}}_i^2 \cdot \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{N}(0,M)} [\boldsymbol{x}_i^2] + \sum_{i\neq j} \widehat{\boldsymbol{w}}_i\widehat{\boldsymbol{w}}_j \cdot \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{N}(0,M)} [\boldsymbol{x}_i\boldsymbol{x}_j]\right] \\
&= \min_{\mathcal{D}\in\Omega} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}}\sim\mathcal{D}} \left[\sum_{i=1}^{d} \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{N}(0,M)} [\boldsymbol{x}_i^2] + \sum_{i\neq j} \widehat{\boldsymbol{w}}_i\widehat{\boldsymbol{w}}_j \cdot \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{N}(0,M)} [\boldsymbol{x}_i\boldsymbol{x}_j]\right] \\
& && (Q = \{-1, 1\})
\end{aligned}$$

$$= \min_{\mathcal{D} \in \Omega} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}} \left[\sum_{i=1}^{d} M_{ii} + \sum_{i \neq j} \widehat{\boldsymbol{w}}_i \widehat{\boldsymbol{w}}_j \cdot M_{ij} \right] \qquad (\mathbf{Cov}(\boldsymbol{x}) = M)$$

We now construct an optimal distribution $\mathcal{D}^*$ for $\mathsf{ADDV}_{\mathcal{N}(0,M)}(0, \{-1, 1\})$. Define

$$\widehat{w}^* := \operatorname*{argmin}_{\widehat{w} \in \{-1,1\}^d} \left[\sum_{i=1}^{d} M_{ii} + \sum_{i \neq j} \widehat{w}_i \widehat{w}_j \cdot M_{ij} \right] = \operatorname*{argmin}_{\widehat{w} \in \{-1,1\}^d} \left[\sum_{i \neq j} \widehat{w}_i \widehat{w}_j \cdot M_{ij} \right]$$

Since the objective is even, let $\mathcal{D}^*$ place equal weight on $\widehat{w}^*$ and $-\widehat{w}^*$ and observe that

$$\begin{aligned} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^*} \left[\sum_{i=1}^{d} M_{ii} + \sum_{i \neq j} \widehat{\boldsymbol{w}}_i \widehat{\boldsymbol{w}}_j M_{ij} \right] &= \sum_{i=1}^{d} M_{ii} + \sum_{i \neq j} \widehat{w}_i^* \widehat{w}_j^* M_{ij} \\ &= \min_{\mathcal{D} \in \Omega} \mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}} \left[\sum_{i=1}^{d} M_{ii} + \sum_{i \neq j} \widehat{\boldsymbol{w}}_i \widehat{\boldsymbol{w}}_j \cdot M_{ij} \right] \end{aligned}$$

by definition of $\widehat{w}^*$. Additionally, for any $x \in \operatorname{supp}(\mathcal{X})$,

$$\mathop{\mathbf{E}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^*} [\langle \widehat{\boldsymbol{w}}, x \rangle] = \langle \widehat{w}^*, x \rangle + \langle -\widehat{w}^*, x \rangle = 0 = \langle w, x \rangle$$

Thus, $\mathcal{D}^*$ is an optimal distribution giving objective value

$$\sum_{i=1}^{d} M_{ii} + \min_{\widehat{w} \in \{-1,1\}^d} \sum_{i \neq j} \widehat{w}_i \widehat{w}_j M_{ij}$$

$\mathcal{D}^*$ may not be the unique optimal distribution, but any optimal distribution must be supported only on vectors $\widehat{w}$ that minimize the quantity $\sum_{i \neq j} \widehat{w}_i \widehat{w}_j M_{ij}$ (since its objective value is the same as $\mathcal{D}^*$). Therefore, given any optimal distribution $\mathcal{D}^{\text{opt}}$, we can extract an arbitrary vector $\widehat{w}^{\text{opt}}$ from its support.

Finally, observe that the max-cut objective on graph G can be written as

$$\text{Max-Cut}(G) := \max_{\widehat{w} \in \{-1,1\}^d} \frac{1}{2} \sum_{i \neq j} \frac{(\widehat{w}_i - \widehat{w}_j)^2}{4} \cdot M_{ij} = \frac{1}{4} \sum_{i \neq j} M_{ij} - \min_{\widehat{w} \in \{-1,1\}^d} \frac{1}{4} \sum_{i \neq j} \widehat{w}_i \widehat{w}_j \cdot M_{ij}$$

or equivalently

$$\text{Max-Cut}(G) := \frac{1}{4} \sum_{i \neq j} M_{ij} - \frac{1}{4} \sum_{i \neq j} \widehat{w}_i^{\text{opt}} \widehat{w}_j^{\text{opt}} M_{ij}$$

Runtime. It is clear that constructing A, y, and M take $O(d^2)$ time. Because computing and reporting $\mathsf{ADDV}_{\mathcal{N}(0,M)}(0, \{-1, 1\})$ is assumed to take polynomial time, extracting w^{opt} from the support of $\mathcal{D}^{\text{opt}}$ also takes polynomial time. Therefore, Algorithm 1 is a polynomial-time reduction from unweighted Max-Cut to ADDV. Because Max-Cut on unweighted, simple graphs is NP-Hard [GJS74], this proves the claim. □

D Average Directional Variance

We begin by making important observations about the ADV objective and defining quantities that will be useful for the algorithms in the remainder of the section. Oftentimes, we also think of w as a multiset.

Furthermore, we assume that the input distribution $\mathcal{X}$ has $\operatorname{supp}(\mathcal{X}) \subseteq \mathbb{R}^d$ and has finite marginal second moments (i.e. $\mathbf{E}_{\boldsymbol{x} \sim \mathcal{X}}[\boldsymbol{x}_i^2]$ is finite for all $i \in [d]$). The problems we consider become trivial otherwise. Our algorithms also assume full-precision access to the marginal second moments.

Lemma D.1. *Let $\mathcal{X}$ be an arbitrary input distribution and $w \in \mathbb{R}^d$ be sorted such that $w_1 \leq \cdots \leq w_d$. Let Q^* be such that $|Q^*| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q^*) = \mathsf{ADV}_{\mathcal{X}}(w, s)$. Then, $w_1, w_d \in Q^*$.*

Lemma D.2. *For every input distribution $\mathcal{X}$, $w \in \mathbb{R}^d$, and $s \in \mathbb{N}$, there exists a quantization set $Q^* \subseteq \mathbb{R}$ such that (i) $|Q^*| = s$, (ii) $Q^* \subseteq w$, and (iii) $\mathsf{ADV}_{\mathcal{X}}(w, Q^*) = \mathsf{ADV}_{\mathcal{X}}(w, s)$.*

We now define and prove a property of the optimization problem that is the crux of all the results in this section. We first define a shorthand, with w indexed so $w_1 \leq \cdots \leq w_d$,

$$C[j,k] \triangleq \sum_{i=j}^{k} \lambda_i \cdot (w_k - w_i)(w_i - w_j)$$

so that $C \in \mathbb{R}^{d \times d}$. We use the standard convention that $C[j,k] = 0$ if $j > k$. Here, $\lambda_i := \mathbf{E}_{\boldsymbol{x} \sim \mathcal{X}}[\boldsymbol{x}_i^2] \in \mathbb{R}^+$ denotes a weight associated with the element w_i. The quantity $C[j,k]$ represents the (weighted) sum of variances of points in the region $[w_j, w_k]$ assuming $w_j, w_k \in Q$.

Lemma D.3 (Lemma 4.2 of [BBBIMV24])**.** *For every input distribution $\mathcal{X}$, Matrix C satisfies the Concave Monge property (Definition A.4).*

Lemma D.4. *For every input distribution $\mathcal{X}$, Matrix C is sorted (Definition A.1).*

The proofs of Lemmas D.1, D.2, D.3, and D.4 are deferred to Appendix H. Note that Lemma D.3 is the weighted version of Lemma 4.2 in [BBBIMV24]. The proof closely follows that of Lemma 4.2 in [BBBIMV24].

Lemma D.5. *Given (unsorted) $w \in \mathbb{R}^d$, with $O(d \log d)$ pre-processing time, any entry of $C[i,j]$ can be computed in $O(1)$ time.*

Proof. Sort w so that $w_1 \leq \cdots \leq w_d$. We define prefix sum vectors

$$\alpha[j] = \sum_{i=1}^{j} \lambda_i \qquad \beta[j] = \sum_{i=1}^{j} \lambda_i w_i \qquad \gamma[j] = \sum_{i=1}^{j} \lambda_i w_i^2$$

Then,

$$\begin{aligned} C[j,k] &:= \sum_{i=j}^{k} \lambda_i \cdot (w_k - w_i)(w_i - w_j) = \sum_{i=j+1}^{k} \lambda_i \cdot (w_k - w_i)(w_i - w_j) \\ &= -w_j w_k \sum_{i=j+1}^{k} \lambda_i + (w_j + w_k) \cdot \sum_{i=j+1}^{k} \lambda_i w_i - \sum_{i=j+1}^{k} \lambda_i w_i^2 \\ &= -w_j w_k \cdot (\alpha[k] - \alpha[j]) + (w_j + w_k) \cdot (\beta[k] - \beta[j]) - (\gamma[k] - \gamma[j]) \end{aligned}$$

Runtime. Sorting w takes $O(d \log d)$ time. Then, the prefix sums α, β, γ can be computed in $O(d)$ time once the vector w is sorted. Thus, the total pre-processing time is $O(d \log d)$. Computing a specific entry $C[j,k]$ can be done with $O(1)$ accesses to w and the prefix sums α, β, γ. □

Remark D.1. Throughout this section, we assume the exact marginal second moments are known (or can computed easily from the representation of the distribution $\mathcal{X}$), i.e. the values λ_i are given for all $i \in [d]$. However, if instead the algorithms are given approximations λ_i' for each λ_i with $\lambda_i \leq \lambda_i' \leq \alpha \lambda_i$, with α known, then the approximation ratios for each of the algorithms in this section increase by a factor of α. The knowledge of α is required for Algorithm 3, in order to be able to bound the search range for the Lagrangian τ.

D.1 Exact Algorithms

In Appendix K of [BBBIMV24], the authors note that optimizing the weighted MSE objective can be solved using a slight variant of the dynamic programming and matrix-search based algorithm presented in their paper. The crux of the improvement in their result comes from the total monotonicity of matrix C that allows a more efficient computation of the dynamic program.

The survey paper of [GLM+17] outlines many results from the 1-dimensional k-means clustering literature, which turn out to immediately give algorithms for the problem of vector quantization with error measured by the weighted MSE objective. For example, it turns out that the exact same dynamic programming and matrix-searching solution of Corollary D.6 of [BBBIMV24] was actually first published by [Wu91], in the setting of 1D k-means. Formally, this implies the following about objective ADV.

Corollary D.6 ([Wu91], Adapted Algorithm 1 of [BBBIMV24]). *There exists an algorithm such that for any $w \in \mathbb{R}^d$, input distribution $\mathcal{X}$, and target quantization set size $s \in \mathbb{N}$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathsf{ADV}_{\mathcal{X}}(w, s)$. The algorithm has runtime $O(d \log d + d \cdot s)$ and uses $O(ds)$ space.*

More generally, there is a spectrum of theoretical results that complete the state-of-the-art optimality profile in differing regimes of k and d. In particular, the results outlined in the survey paper of [GLM+17] can be applied to any problem whose structure can be shown to satisfy the Concave Monge property (Definition A.4) (including but not limited to, 1-dimensional k-means clustering, 1-dimensional k-medians clustering, and $\mathsf{ADV}_{\mathcal{X}}$).

More specifically, the algorithms outlined in [GLM+17] solve the following problem. Let G be a complete, directed acyclic graph on d vertices with weights $A \in \mathbb{R}^{d \times d}$ that satisfy the Concave Monge property (Definition A.4). Then, given any two vertices $i, j \in [d]$ and integer k, we wish to find the minimum weight length-k path between i and j in G.

[AST93] gives (i) an algorithm with runtime $O(d\sqrt{k \log d})$, and (ii) an algorithm with runtime $O(d \log \Delta)$ where $\Delta := \max_{i,j} A[i,j] - \min_{i,j} A[i,j]$. When $k = \Omega(\log d)$, [Sch98] gives an algorithm with runtime $d2^{O(\sqrt{\log \log d \log k})}$. At a high level, these algorithms work by solving the following *regularized* version of the problem

$$\min_{\ell, \{i_1, \ldots, i_\ell\} \subseteq [d]} \sum_{j=1}^{\ell-1} C[i_j, i_{j+1}] + \tau \ell$$

where the task is to find an integer ℓ and a path of length ℓ that minimizes the regularized objective function (i.e. each edge has additional cost τ). In doing so, the results of [AST93, Sch98] use ideas and results from [Wil88] that solves the unregularized version (still assuming the weights are Concave Monge).

Notice that by Lemma D.3, applying these algorithms to weights $C[i,j]$ directly gives a solution optimizing $\mathsf{ADV}_{\mathcal{X}}$, implying the following results.

Corollary D.7 ([AST93]). *There exist algorithms such that for any $w \in \mathbb{R}^d$, input distribution $\mathcal{X}$, and target quantization set size $s \in \mathbb{N}$, return a quantization set Q such that $|Q| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathsf{ADV}_{\mathcal{X}}(w, s)$. The algorithms have runtime (i) $O(d\sqrt{s \log d})$ and (ii) $O(d \log \Delta)$ where $\Delta := \max_{i,j} C[i,j] - \min_{i,j} C[i,j]$.*

Corollary D.8 ([Sch98]). *There exists an algorithm such that for any $w \in \mathbb{R}^d$, input distribution $\mathcal{X}$, and target quantization set size $s = \Omega(\log d)$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathsf{ADV}_{\mathcal{X}}(w, s)$. The algorithm has runtime $d2^{O(\sqrt{\log \log d \log s})}$.*

This summarizes the state-of-the-art of exact algorithms for $\mathsf{ADV}_{\mathcal{X}}$.

D.2 Approximation Algorithms

The work of [BBBIMV24] provides a bi-criteria approximation algorithm for the *unweighted* MSE objective. In particular, their algorithm returns a quantization set of size $2s - 2$ whose quality is an additive approximation to that of the optimal. It is easy to see, following their analysis, that the same algorithm, analyzed under the *weighted* MSE objective results in the following weak additive approximation guarantee.

Corollary D.9 (Lemma 6.1 of [BBBIMV24]). *There exist algorithms such that for any $w \in \mathbb{R}^d$, input distribution $\mathcal{X}$, and target quantization set size $s \in \mathbb{N}$, returns a quantization set Q such that $|Q| = 2s-2$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) \leq \mathsf{ADV}_{\mathcal{X}}(w, s) + \sum_{i=1}^{d} \lambda_i \cdot \Delta^2 / m^2$. Here, $\Delta := \max_{i,j \in [d]} w_i - w_j$. The algorithm has runtime $O(d + ms)$.*

In this section, we develop approximation algorithms for optimizing the $\mathsf{ADV}_{\mathcal{X}}$ (weighted MSE) objective. We first give an s-approximation, then improve it to a $(1+\varepsilon)$-approximation. Along the way, we observe that the main ideas from solutions to MDV and $\mathsf{ADV}_{\mathcal{X}}$ (in particular, building data-dependent coresets) can be used to develop a much stronger approximation algorithm for quantization under the unweighted MSE objective. That result is detailed in Appendix E.

D.2.1 An s-approximation

Here, we give an s-approximation algorithm by defining a new intermediate objective function that (loosely speaking) "interpolates" between MDV and $\mathsf{ADV}_{\mathcal{X}}$. We then show that the optimum of this new *mixed* objective is an s-approximation to that of $\mathsf{ADV}_{\mathcal{X}}$. In particular, consider a quantization set $Q = \{w_{i_1}, \ldots, w_{i_{|Q|}}\} \subseteq w$ where $w_{i_1} \leq \ldots \leq w_{i_{|Q|}}$.

We now define the objective

$$\mathsf{MixDV}_{\mathcal{X}}(w, s) := \min_{Q \subset w : |Q| \leq s} \max_{j \in [s-1]} C[i_j, i_{j+1}] \tag{2}$$

which seeks to find the quantization set Q (lying entirely inside w) that minimizes the maximum sum of the variances in the intervals between adjacent quantization points of Q. We also define a shorthand to denote the objective value of a fixed quantization set Q:

$$\mathsf{MixDV}_{\mathcal{X}}(w, Q) := \max_{j \in [|Q|-1]} C[i_j, i_{j+1}]$$

We now prove the following relationship between $\mathsf{MixDV}_{\mathcal{X}}$ and $\mathsf{ADV}_{\mathcal{X}}$.

Lemma D.10. *For $s \geq 2$, $\mathsf{ADV}_{\mathcal{X}}(w, s)/s \leq \mathsf{MixDV}_{\mathcal{X}}(w, s) \leq \mathsf{ADV}_{\mathcal{X}}(w, s)$*

Proof. Notice that for any quantization set $Q = \{w_{i_1}, \ldots, w_{i_s}\} \subseteq w$ of size s, it is the case that

$$\mathsf{ADV}_{\mathcal{X}}(w, s) \leq \mathsf{ADV}_{\mathcal{X}}(w, Q) = \sum_{j \in [s-1]} C[i_j, i_{j+1}] \leq |Q| \cdot \mathsf{MixDV}_{\mathcal{X}}(w, Q)$$

where all steps follow by the definitions of $\mathsf{ADV}_{\mathcal{X}}$ and $\mathsf{MixDV}_{\mathcal{X}}$. Let $Q^*_{\mathsf{MixDV}_{\mathcal{X}}} \subseteq w$ be such that $|Q^*_{\mathsf{MixDV}_{\mathcal{X}}}| = s$ and $\mathsf{MixDV}_{\mathcal{X}}(w, Q^*_{\mathsf{MixDV}_{\mathcal{X}}}) = \mathsf{MixDV}_{\mathcal{X}}(w, s)$. Then, instantiating the above with $Q^*_{\mathsf{MixDV}_{\mathcal{X}}}$, we conclude that $\mathsf{MixDV}_{\mathcal{X}}(w, s) \geq \mathsf{ADV}_{\mathcal{X}}(w, s)/s$. It remains to show that $\mathsf{MixDV}_{\mathcal{X}}(w, s) \leq \mathsf{ADV}_{\mathcal{X}}(w, s)$. Let $Q^*_{\mathsf{ADV}_{\mathcal{X}}}$ be such that $|Q^*_{\mathsf{ADV}_{\mathcal{X}}}| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q^*_{\mathsf{ADV}}) = \mathsf{ADV}_{\mathcal{X}}(w, s)$. It is clear that

$$\mathsf{MixDV}_{\mathcal{X}}(w, s) \leq \mathsf{MixDV}_{\mathcal{X}}(w, Q^*_{\mathsf{ADV}}) \leq \mathsf{ADV}_{\mathcal{X}}(w, Q^*_{\mathsf{ADV}}) = \mathsf{ADV}_{\mathcal{X}}(w, s)$$

where again all steps follow by definition of the objectives. □

Lemma D.10 suggests a natural approximation algorithm for $\mathsf{ADV}_{\mathcal{X}}$. In particular, an exact solution to $\mathsf{MixDV}_{\mathcal{X}}$ immediately gives an s-approximation to $\mathsf{ADV}_{\mathcal{X}}$.

Theorem 3. *There exists an algorithm such that for any $w \in \mathbb{R}^d$, input distribution $\mathcal{X}$, and target quantization set size $s \in \mathbb{N}$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{MixDV}_{\mathcal{X}}(w, Q) = \mathsf{MixDV}_{\mathcal{X}}(w, s)$, and consequently $\mathsf{ADV}_{\mathcal{X}}(w, s)/s \leq \mathsf{MixDV}_{\mathcal{X}}(w, Q) \leq \mathsf{ADV}_{\mathcal{X}}(w, s)$. The runtime of the algorithm is $O(d \log d)$.*

Proof. To produce an s-approximation to ADV without relaxing the target quantization set size, we will use Algorithm 2 to exactly solve the $\mathsf{MixDV}_{\mathcal{X}}$ objective.

Observe that the true optimal cost $\mathsf{MixDV}_{\mathcal{X}}(w, s)$ is of the form $C[i_j, i_k]$ for some $j, k \in [d]$. Algorithm 2 binary searches over the $O(d^2)$ many possible objective values, using the linear time checking algorithm `Sorted-Selection` from Lemma A.1 to direct the binary search. The fact that matrix C is sorted is proven as Lemma D.4 in Section H. It remains to prove the correctness of the checking step (Lines 8 - 17). A given objective value $v \in \mathbb{R}$ is achievable if there exists an arrangement of quantization points $Q \subset w$ for $\mathsf{MixDV}_{\mathcal{X}}(w, Q) \leq v$. Algorithm 2 greedily constructs a quantization set by performing a linear scan through the sorted w and only adding a quantization point once the sum of variances exceeds v. It is trivial to see that if $\mathsf{MixDV}_{\mathcal{X}}(w, s) \leq v$, then the greedily constructed quantization set Q_{Greedy} must have $|Q_{\text{Greedy}}| \leq s$ and $\mathsf{MixDV}_{\mathcal{X}}(w, Q_{\text{Greedy}}) \leq v$.

By construction, Q_{Greedy} is such that $\mathsf{MixDV}_{\mathcal{X}}(w, Q_{\text{Greedy}}) \leq v$. Therefore, if $|Q_{\text{Greedy}}| \leq s$, objective value v is achievable. Otherwise, it is not. This check in lines 13 - 17 directs the binary search. It follows that the returned quantization set Q^* satisfies $\mathsf{MixDV}_{\mathcal{X}}(w, Q^*) = \mathsf{MixDV}_{\mathcal{X}}(w, s)$. If $|Q^*| < s$, we pad it with arbitrary elements of w; this only splits intervals, which by Lemma D.4 does not increase the cost of any interval, and so leaves $\mathsf{MixDV}_{\mathcal{X}}(w, Q^*)$ unchanged. The claim that $\mathsf{ADV}_{\mathcal{X}}(w, s)/s \leq \mathsf{MixDV}_{\mathcal{X}}(w, Q^*) \leq \mathsf{ADV}_{\mathcal{X}}(w, s)$ then follows from Lemma D.10.

Algorithm 2: Exact Algorithm for $\mathsf{MixDV}_{\mathcal{X}}$

Input: $w \in \mathbb{R}^d$, $s \in \mathbb{N}$, $\lambda \in \mathbb{R}^d$ where $\lambda_i = \mathbf{E}_{\boldsymbol{x} \sim \mathcal{X}}[\boldsymbol{x}_i^2]$ for $i \in [d]$
Output: Quantization set Q

```
1  Sort w and compute prefix sums α, β, γ
2  Initialize k↑ ← d² and k↓ ← 0
3  Initialize Q* ← ∅
4  while k↑ ≥ k↓ do
5      k ← ⌊(k↑ + k↓)/2⌋
6      v ← Sorted-Selection(α, β, γ, k)
7      ▷ Check whether objective value v is possible with s quantization points
8      Initialize Q ← {w_1, w_d} and i ← 1
9      for j = 2, ..., d do
10         if C[i, j] > v then
11             Q ← Q ∪ {w_{j−1}}
12             Set i ← j − 1
13     if |Q| ≤ s then
14         k↑ ← k − 1
15         Q* ← Q
16     else
17         k↓ ← k + 1
18 return Q*
```

Runtime. Sorting w takes $O(d \log d)$ time. Computing the prefix sums α, β, γ takes $O(d \log d)$ time as per Lemma D.5. The while loop is performing binary search over $O(d^2)$ many values and is thus only run $O(\log d)$ times. The k-selection algorithm takes time $O(d)$ as per Lemma A.1 (note that the matrix C is never explicitly written out by the algorithm. It is implicitly defined by the prefix sums α, β, γ which allow $O(1)$ access time to an entry of C as per Lemma D.5). The *check* logic in Lines 8 - 17 take $O(d)$ time. Therefore, the total algorithm runtime is $O(d \log d)$. □

D.2.2 A $(1+\varepsilon)$-approximation

Using the s-approximation from Algorithm 2, we can get a $(1+\varepsilon)$-approximation algorithm with efficient runtime. At a high level, the idea is to first obtain an s-approximation to the objective value, and then use it to bound both the range of the search for the Lagrangian multiplier and the precision to which it must be found in the $O(d \log \Delta)$ time algorithm of [AST93] (Corollary D.7). We begin by giving a primer on this algorithm. This discussion closely mirrors that in Section 3 of [GLM+17]. Recall that the problem at hand is to optimize the *regularized* objective

$$\min_{\ell, \{i_1, \ldots, i_\ell\} \subseteq [d]} \sum_{j=1}^{\ell-1} C[i_j, i_{j+1}] + \tau \ell$$

which corresponds to choosing the lightest path in a complete DAG with weights given by C and cost τ for each node in the path. Observe that the unregularized version is simply optimizing $\mathsf{ADV}_{\mathcal{X}}$. Setting $\tau = 0$ means that there is no cost associated with choosing extra nodes, so the optimal path is to include every node, resulting in an objective value of 0. Now consider setting $\tau \geq \mathrm{opt}_{d-1}$ where

$$\mathrm{opt}_{d-1} = \min_{\{i_1, \ldots, i_{d-1}\} \subseteq [d]} \sum_{j=1}^{\ell-1} C[i_j, i_{j+1}]$$

is equivalent to $\mathsf{ADV}_{\mathcal{X}}(w, d-1)$. For these values of τ, it is better to construct the optimal $d-1$ length path than construct a d length path. This is because the extra additive τ cost of using the dth node outweighs the reduction of $C[i, j]$ costs when using d nodes. In this way, we find *critical points* for the setting of τ at which the optimal path is of length k. That is $\tau_k = \mathrm{opt}_k - \mathrm{opt}_{k+1}$ ensures that the optimal path is of length k. It turns out that $0 = \tau_d \leq \tau_{d-1} \leq \ldots \leq \tau_1$ (see [AST93, GLM+17] for proofs and details). The $O(d \log \Delta)$ algorithm then proceeds by binary searching for the correct

setting of τ; it checks whether a particular setting of τ yields an optimal solution of length s using the $O(d)$ time algorithm of [AST93, Sch98] which finds shortest paths on DAGs with concave monge weights. In particular, [AST93, Sch98] gives three types of algorithms: (1) `ShortestPathDAG-Min` which returns the shortest path with optimal cost (2) `ShortestPathDAG-Max` which returns the longest path with optimal cost and (3) `ShortestPathDAG`(k) which returns the length-k path with optimal cost (if it exists). All have runtime $O(d)$. Running and checking these algorithms allows us to direct the binary search.

Algorithm 3 first runs the s-approximation Algorithm 2 and computes the objective value $v = \mathsf{MixDV}_{\mathcal{X}}(w, s)$ of the returned quantization set, which by Lemma D.10 satisfies $\mathsf{ADV}_{\mathcal{X}}(w, s)/s \le v \le \mathsf{ADV}_{\mathcal{X}}(w, s)$ (this value can be computed from the returned quantization set in linear time trivially). It then performs the binary search for τ described above, but only over multiples of $\varepsilon v/s$. Each iteration runs `ShortestPathDAG-Min` and `ShortestPathDAG-Max` on the current setting of τ and records the lengths k_{min} and k_{max} of the two returned paths. If some multiple of $\varepsilon v/s$ admits an optimal path of length s, that path is returned. Otherwise, the search ends at two consecutive multiples of $\varepsilon v/s$, where all optimal paths for the smaller value are longer than s and all optimal paths for the larger value are shorter than s. We then combine an optimal path for each of these two values into a single path of length s, using the exchange argument of Lemma D.11 below.

Throughout, a *path* is an increasing sequence of indices $P = (i_1, \dots, i_k)$ with $i_1 = 1$ and $i_k = d$. We write $|P| = k$ for its length (i.e. its number of nodes) and $\mathrm{Cost}(P) = \sum_{j=1}^{k-1} C[i_j, i_{j+1}]$ for its cost, and opt_k for the minimum cost of a path of length k. Recall that by Lemmas D.1 and D.2, a path P corresponds to the quantization set $Q = \{w_{i_1}, \dots, w_{i_k}\}$ with $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathrm{Cost}(P)$, and that $\mathrm{opt}_s = \mathsf{ADV}_{\mathcal{X}}(w, s)$. For $\mu \ge 0$, we say that P is *μ-optimal* if it minimizes $\mathrm{Cost}(P) + \mu|P|$ over all paths. Since all paths of the same length incur the same penalty, a μ-optimal path P satisfies $\mathrm{Cost}(P) = \mathrm{opt}_{|P|}$.

Algorithm 3: $(1+\varepsilon)$-Approximation Algorithm for $\mathsf{ADV}_{\mathcal{X}}$

Input: $w \in \mathbb{R}^d$, $s \in \mathbb{N}$, $\varepsilon > 0$, $\lambda \in \mathbb{R}^d$ where $\lambda_i = \mathbf{E}_{\boldsymbol{x} \sim \mathcal{X}}[\boldsymbol{x}_i^2]$ for $i \in [d]$
Output: Quantization set Q

$v \leftarrow \mathsf{MixDV}_{\mathcal{X}}(w, \text{Algorithm } 2(w, s, \lambda))$
Sort w and compute prefix sums α, β, γ
Initialize $\tau^{\uparrow} \leftarrow \lceil s^2(1+\varepsilon)/\varepsilon \rceil$ and $\tau^{\downarrow} \leftarrow 0$
while $\tau^{\uparrow} \ge \tau^{\downarrow}$ **do**
  $\tau \leftarrow \lfloor (\tau^{\uparrow} + \tau^{\downarrow})/2 \rfloor$
  $k_{\mathsf{min}} \leftarrow$ `ShortestPathDAG-Min`$(C, \tau\varepsilon v/s)$
  $k_{\mathsf{max}} \leftarrow$ `ShortestPathDAG-Max`$(C, \tau\varepsilon v/s)$
  if $k_{min} \le s \le k_{max}$ **then**
    Return `ShortestPathDAG`$(C, \tau\varepsilon v/s, s)$
  else if $s < k_{min}$ **then**
    $\tau^{\downarrow} \leftarrow \tau + 1$
  else
    $\tau^{\uparrow} \leftarrow \tau - 1$
Return the path of Lemma D.11 for `ShortestPathDAG-Min`$(C, \tau^{\uparrow}\varepsilon v/s)$ and `ShortestPathDAG-Max`$(C, \tau^{\downarrow}\varepsilon v/s)$ ▷ here $\tau^{\downarrow} = \tau^{\uparrow} + 1$

Lemma D.11. *Let $P_1 = (a_1, \dots, a_{k_1})$ and $P_2 = (b_1, \dots, b_{k_2})$ be paths with $k_2 < s < k_1$. Then, there exist paths P and P' with $|P| = s$ and $|P'| = k_1 + k_2 - s$ such that*

$$\mathrm{Cost}(P) + \mathrm{Cost}(P') \le \mathrm{Cost}(P_1) + \mathrm{Cost}(P_2)$$

Moreover, P can be computed in $O(d)$ time.

Proof. Let $t = s - k_2 \ge 1$ and notice that $q + t \le s < k_1$ for every $q \in [k_2]$. Since $b_1 = 1 \le a_{1+t}$ while $b_{k_2} = d = a_{k_1} > a_{k_2+t}$, taking $q \in [k_2 - 1]$ to be minimal with $b_{q+1} > a_{q+t+1}$ also gives $b_q \le a_{q+t}$. Setting $p = q + t$, we therefore have $b_q \le a_p < a_{p+1} < b_{q+1}$. Define

$$P := (a_1, \dots, a_p, b_{q+1}, \dots, b_{k_2}) \qquad \text{and} \qquad P' := (b_1, \dots, b_q, a_{p+1}, \dots, a_{k_1})$$

Both are paths, of lengths $p + k_2 - q = s$ and $q + k_1 - p = k_1 + k_2 - s$ respectively. Their edges are exactly those of P_1 and P_2, except that (a_p, a_{p+1}) and (b_q, b_{q+1}) are replaced by (a_p, b_{q+1}) and (b_q, a_{p+1}). Since $b_q \le a_p \le a_{p+1} \le b_{q+1}$ and C satisfies the Quadrangle Inequality (Definition A.5; see the proof of Lemma D.3 in Appendix H),

$$C[b_q, a_{p+1}] + C[a_p, b_{q+1}] \le C[b_q, b_{q+1}] + C[a_p, a_{p+1}]$$

which proves the claim. Finding q requires a single pass over P_2, so P can be computed in $O(d)$ time. □

Theorem 4. *There exists an algorithm that for any given vector $w \in \mathbb{R}^d$, target quantization set size $s \in \mathbb{N}$, input distribution $\mathcal{X}$, and $\varepsilon > 0$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) \le (1+\varepsilon) \cdot \mathsf{ADV}_{\mathcal{X}}(w, s)$. The runtime of the algorithm is $O(d \log(d/\varepsilon))$.*

Proof. The algorithm is described in Algorithm 3. Let $\eta = \varepsilon v / s$ and $T = \lceil s^2(1+\varepsilon)/\varepsilon \rceil$. By Theorem 3, Line 1 computes $v = \mathsf{MixDV}_{\mathcal{X}}(w, s)$, which by Lemma D.10 satisfies $\mathrm{opt}_s / s \le v \le \mathrm{opt}_s$.

Suppose first that the algorithm returns in Line 9, and let τ denote the value of the loop variable in that iteration. Since $k_{\min} \le s \le k_{\max}$, the call to `ShortestPathDAG` in that line returns a $\tau\eta$-optimal path of length s [AST93, Sch98], and so its cost is $\mathrm{opt}_s = \mathsf{ADV}_{\mathcal{X}}(w, s)$, as desired.

Call an integer τ *too small* if `ShortestPathDAG-Min`$(C, \tau\eta)$ returns a path of length greater than s, and *too large* if `ShortestPathDAG-Max`$(C, \tau\eta)$ returns a path of length less than s. Lines 10 - 13 maintain that $\tau^\downarrow - 1$ is too small whenever $\tau^\downarrow > 0$ and that $\tau^\uparrow + 1$ is too large whenever $\tau^\uparrow < T = \lceil s^2(1+\varepsilon)/\varepsilon \rceil$, and the loop terminates with $\tau^\downarrow = \tau^\uparrow + 1$. Observe that 0 is not too large, since C is nonnegative and $C[i, i+1] = 0$ for all i, so the path through all d nodes is 0-optimal. Observe also that T is not too small: a $T\eta$-optimal path P satisfies $\mathrm{Cost}(P) + T\eta|P| \le \mathrm{opt}_s + T\eta s$, which for $|P| > s$ would give $T\eta \le \mathrm{opt}_s$, but $T\eta \ge (1+\varepsilon)sv > \mathrm{opt}_s$ (Lemma D.10). It follows that $0 \le \tau^\uparrow < T$ at termination, and so $\tau^\uparrow$ is too small while $\tau^\downarrow$ is too large.

Let P_1 and P_2 be the paths computed in Line 14, so that $k_1 := |P_1| > s$ and $k_2 := |P_2| < s$, and let P and P' be the paths given by Lemma D.11, with $|P| = s$ and $|P'| = m := k_1 + k_2 - s$. Since $\mathrm{Cost}(P_1) = \mathrm{opt}_{k_1}$, $\mathrm{Cost}(P_2) = \mathrm{opt}_{k_2}$, and $\mathrm{Cost}(P') \ge \mathrm{opt}_m$, Lemma D.11 gives

$$\mathrm{Cost}(P) \le \mathrm{opt}_{k_1} + \mathrm{opt}_{k_2} - \mathrm{opt}_m. \tag{3}$$

As P_1 is $\tau^\uparrow\eta$-optimal and P_2 is $\tau^\downarrow\eta$-optimal,

$$\mathrm{opt}_{k_1} + \tau^\uparrow \eta\, k_1 \le \mathrm{opt}_m + \tau^\uparrow \eta\, m \qquad \text{and} \qquad \mathrm{opt}_{k_2} + \tau^\downarrow \eta\, k_2 \le \mathrm{opt}_s + \tau^\downarrow \eta\, s.$$

Rearranging each of these as $\mathrm{opt}_{k_1} - \mathrm{opt}_m \le \tau^\uparrow\eta\,(m - k_1)$ and $\mathrm{opt}_{k_2} - \mathrm{opt}_s \le \tau^\downarrow\eta\,(s - k_2)$, then adding and using $m - k_1 = k_2 - s$ and $\tau^\downarrow = \tau^\uparrow + 1$, we obtain

$$\begin{aligned} \mathrm{opt}_{k_1} + \mathrm{opt}_{k_2} - \mathrm{opt}_m - \mathrm{opt}_s &\le \tau^\uparrow\eta\,(m - k_1) + \tau^\downarrow\eta\,(s - k_2) \\ &= (\tau^\downarrow - \tau^\uparrow) \cdot \eta\,(s - k_2) \\ &= \eta\,(s - k_2) < \eta s = \varepsilon v. \end{aligned}$$

Thus,

$$\mathrm{Cost}(P) \le \mathrm{opt}_{k_1} + \mathrm{opt}_{k_2} - \mathrm{opt}_m < \mathrm{opt}_s + \varepsilon v \le (1+\varepsilon)\mathrm{opt}_s.$$

and so the quantization set Q corresponding to P satisfies $|Q| = s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathrm{Cost}(P) < (1+\varepsilon) \cdot \mathsf{ADV}_{\mathcal{X}}(w, s)$.

Runtime. Running Algorithm 2 and computing the objective value v takes $O(d \log d)$ time by Theorem 3. Sorting w and computing the prefix sums takes $O(d \log d)$ time as well by Lemma D.5. The binary search (lines 4 - 13) searches over $O(s^2/\varepsilon)$ many values. Each iteration of the binary search runs a `ShortestPathDAG` algorithm $O(1)$ times, each of which takes $O(d)$ time. Therefore, the entire binary search portion of Algorithm 3 has runtime $O(d \log(s/\varepsilon))$. Line 14 makes two further such calls and applies Lemma D.11 in $O(d)$ time. The total runtime of Algorithm 3 is then $O(d \log d + d \log(s/\varepsilon)) = O(d \log(d/\varepsilon))$. □

D.3 Practical Algorithms

Our practical algorithms build off the `Wilber` algorithm of [GLM+17] (and our implementations build off their open-source code as well). The `Wilber` algorithm proceeds by searching for a correct Lagrangian multiplier τ on which to solve a relaxed (or *regularized*) version of the problem; see Section D.1.

To improve the runtime of vanilla `Wilber`, which searches for τ using a method the authors of [GLM+17] call interpolation search, we reduce the search space needed by employing approximation algorithms to get a rough estimate of the cost.

In particular, [GLM+17] show that the optimal value of τ occurs at $\tau = \mathrm{opt}_s - \mathrm{opt}_{s+1}$, where opt_s is the cost (of the optimal ADV solution) with s quantization values and opt_{s+1} the cost with $s+1$. This immediately suggests two ways of reducing the search space and thus runtime of the algorithm:

1. Run a fast, but potentially looser, approximation algorithm, and use this to upper bound τ at some value $U = \mathsf{ADV}(w, Q)$, where w is the input vector and Q is the set returned by the approximation. Then, search the range $[0, U]$ for τ. This is the *fast approximation* technique.
2. Run a slower, but more accurate, approximation algorithm and obtain estimates $\widehat{v}_s, \widehat{v}_{s+1}$ for $\mathrm{opt}_s, \mathrm{opt}_{s+1}$, respectively. Then, search the range $[(\widehat{v}_s - \widehat{v}_{s+1})/2, 2(\widehat{v} - \widehat{v}_{s+1})]$ (where the factors 2 are parameters which can be increased or decreased depending on the quality of the approximation). This is the *accurate approximation* technique.

When using the fast approximation technique, we always construct Q as the optimal solution for $\mathsf{MixDV}_{\mathcal{X}}$ (see Section D.2.1), as it is very fast to compute and provably an s-approximation (Lemma D.10). The hope of using the accurate approximation technique is that the additional runtime of the approximation algorithm will be offset by a much smaller search space; for this we use both a faster implementation of approximation `QUIVER` (as discussed in Section D.3.1) and a new approximation algorithm `MixApprox`.

`MixApprox` proceeds similarly to approximate `QUIVER`: given a parameter m, we construct a subset $\mathcal{C} \subseteq w$ of size m and find the optimal s values to select from this subset $\mathcal{C}$.[11] The difference lies in how C is constructed: while approximate `QUIVER` uniformly spaces C across the range $[\min(w), \max(w)]$, in `MixApprox`, we set $\mathcal{C}$ to be the optimal solution to $\mathsf{MixDV}_{\mathcal{X}}$ over w with m quantization points. This *adaptive* approach to constructing $\mathcal{C}$ gives much better approximations, allowing us to use a smaller value of m yet still obtain comparable approximation ratio. In the accurate approximation technique, the approximation algorithm is only run once, while an exact solver is run twice on the set returned by the approximation algorithm (to estimate opt_s and opt_{s+1}), so a smaller value of m reduces the runtime of these calls.

We compare the fast approximation technique with the accurate approximation technique, using both `MixApprox` and a faster implementation of approximate `QUIVER`. Due to the robust nature of `MixApprox` and the ability to run with smaller value m, using `MixApprox` as the approximation algorithm gives the most reliable performance across data distributions (see the supplementary information for a CSV file of performance evaluations over a range of distributions). However, there is still a cost of running the algorithm for $\mathsf{MixDV}_{\mathcal{X}}$ rather than simply uniformly spacing points, so we expect on some data, especially those which are quite uniform, using (improved) approximate `QUIVER` as the estimator will lead to better performance.

D.3.1 Improved Approximate QUIVER

Using our exact algorithms, we are able to also speed up approximate `QUIVER` from [BBBIMV24]. Usually, m is not too much larger than s, and so for this special case, using the accurate approximation technique described above is not faster than the fast approximation technique. So, we warm-start `Wilber` with the solution cost of $\mathsf{MixDV}_{\mathcal{X}}$; see Figure 4 for a runtime comparsion.

One may be interested in using `MixApprox` as an approximation algorithm, since it can obtain very good approximations with small values of m. Unfortunately, the cost of computing $\mathsf{MixDV}_{\mathcal{X}}$ usually negates the cost of increasing m, except on very skewed data. This is in contrast to our exact

[11] When solving on this subset, we do not recurse, and instead always employ the fast approximation technique with the estimate from $\mathsf{MixDV}_{\mathcal{X}}$.

Table 3: Runtime of `Wilber` in milliseconds (ms) with different search methods, across vectors of size d sampled from LogNormal$(0, 1)$ and $s = 64$, averaged across 10 trials. Interp. search is the standard interpolation search method outlined in [GLM+17], MixDV$_{\mathcal{X}}$ is the fast approximation technique with solving MixDV$_{\mathcal{X}}$, `MixApprox` is the accurate approximation technique using `MixApprox`, and Imp. approx `QUIVER` is the accurate approximation technique using our improved implementation of approximate `QUIVER`. For `MixApprox`, we use $m = 4s$, and for approximate `QUIVER`, we use $m = 200s$.

d	Interp. Search	MixDV$_{\mathcal{X}}$	`MixApprox`	Imp. Apx. `QUIVER`
100,000	70.27	72.41	42.49	43.67
200,000	143.33	144.57	68.63	82.84
300,000	221.92	205.69	100.32	114.77
400,000	295.17	328.10	175.08	169.40
500,000	371.57	378.40	182.93	180.73
600,000	428.94	474.88	241.17	236.43
700,000	389.62	521.06	275.22	270.25
800,000	650.12	622.00	297.00	301.70
900,000	773.60	663.57	327.45	332.53
1,000,000	906.50	780.26	349.30	402.50

algorithms, where the additional call of the exact solver (solving for not just a quantization set of size s but also one of size $s + 1$) allows `MixApprox` to be the more performant choice.

E An Improved Approximation Algorithm for Unweighted MSE

Alongside the exact algorithm for weighted (and therefore also unweighted) MSE given by Corollary D.6, [BBBIMV24] also present the following approximation algorithm for *unweighted* MSE.

Corollary E.1 (Lemma 6.1 of [BBBIMV24])**.** *There exists an algorithm with runtime $O(d + ms)$ that returns a quantization set Q such that $|Q| = 2s - 2$ and $\mathsf{MSE}(w, Q) \leq \mathsf{MSE}(w, s) + d\Delta^2/m^2$. Here, $\Delta := \max_{i,j\in[d]} w_i - w_j$.*

The approximation algorithm of Corollary E.1 leaves much to be desired. It gives a bicriteria approximation: relaxing the reported quantization set size while guaranteeing only a weak additive approximation to the optimal mean squared error. In this section, we outline a new approximation algorithm for optimizing unweighted MSE under the standard stochastic quantization rounding distribution. We give a new *data-dependent* bicriteria algorithm with stronger approximation guarantees than that of Corollary E.1.

E.1 An Improved Bicriteria Approximation Algorithm

At a high level the bicriteria approximation algorithm of Lemma E.1 works by considering a candidate set of quantization points given by m points uniformly spaced along the range of w. It then solves exactly on this candidate set to choose the best $2s - 2$ points. Algorithm 4 improves upon this by choosing the candidate quantization points in a data-dependent fashion. We employ the s-center clustering algorithm of Lemma A.3 to construct the candidate *coreset*, subdivide these clustered regions evenly, then exactly solve on this new candidate set.

The correctness of Algorithm 4 relies on a connection between the unweighted MSE and maximum variance when rounding according to the standard stochastic quantization distribution. In particular, let $s \in \mathbb{N}$ be the target quantization set size, and consider quantization set Q_{MSE} of size s that minimizes the mean-squared error

$$\mathsf{MSE}(\widehat{\boldsymbol{w}}, w) = \mathbf{E}\left[\|\widehat{\boldsymbol{w}} - w\|_2^2\right] = \sum_{i\in[d]} \mathbf{Var}[\widehat{\boldsymbol{w}}_i].$$

Then, consider quantization set Q_{MaxVar} of size s that minimizes $\max_{i\in[d]} \mathbf{Var}[\widehat{\boldsymbol{w}}_i]$. It is immediate that

$$\max_{i\in[d]} \underset{\widehat{\boldsymbol{w}}\sim\mathcal{D}^{\mathsf{SSQ}}(w,Q_{\mathsf{MaxVar}})}{\mathbf{Var}}[\widehat{\boldsymbol{w}}_i] \leq \sum_{i\in[d]} \underset{\widehat{\boldsymbol{w}}\sim\mathcal{D}^{\mathsf{SSQ}}(w,Q_{\mathsf{MSE}})}{\mathbf{Var}}[\widehat{\boldsymbol{w}}_i]$$

Recall that this can simply be written as $\mathsf{MDV}(w,s) \le \mathsf{MSE}(w,s)$. Concretely, this relation will allow us to use a "worst-case" clustering algorithm (s-center clustering) and connect back to the unweighted MSE.

We note that Algorithms 2 and 3 for $\mathsf{MixDV}_{\mathcal{X}}$ and $\mathsf{ADV}_{\mathcal{X}}$ can be seen to work more generally for a candidate set of quantization points $P = \{p_1, \dots, p_m\} \subset \mathbb{R}$ (such that $p_1 \le w_1$ and $p_m \ge w_m$).[12] In particular, the modified objectives are

$$\mathsf{MixDV}_{\mathcal{X}}(w,s) := \min_{Q \subset P : |Q| \le s} \max_{j \in [s-1]} C[i_j, i_{j+1}]$$

$$\mathsf{ADV}_{\mathcal{X}}(w,s) := \min_{Q \subset P : |Q| \le s} \sum_{j \in [s-1]} C[i_j, i_{j+1}]$$

where $Q = \{p_{i_1}, \dots, p_{i_{|Q|}}\}$ and $C[i_j, i_{j+1}] := \sum_{w_i \in [p_{i_j}, p_{i_{j+1}}]} (p_{i_{j+1}} - w_i)(w_i - p_{i_j})$. Given P and $O(|P| \log |P| + d \log s)$ preprocessing time, we can provide $O(1)$ time access to matrix C. In particular, we wish to compute prefix sums $\alpha, \beta, \gamma \in \mathbb{R}^{|P|}$

$$\alpha[j] = \sum_{w_i \in [p_1, p_j]} 1 \qquad \beta[j] = \sum_{w_i \in [p_1, p_j]} w_i \qquad \gamma[j] = \sum_{w_i \in [p_1, p_j]} w_i^2$$

This can be achieved by first sorting P, then for each $i \in [d]$ finding the unique $j \in [|P|]$ for which $w_i \in [p_j, p_{j+1}]$ (which be done in $O(d \log |P|)$ time using binary search). We then construct A, B, Γ in each bucket

$$A[j] = \sum_{w_i \in [p_j, p_{j+1}]} 1 \qquad B[j] = \sum_{w_i \in [p_j, p_{j+1}]} w_i \qquad \Gamma[j] = \sum_{w_i \in [p_j, p_{j+1}]} w_i^2$$

Then,

$$\alpha[j] = \sum_{i=1}^{j-1} A[i] \qquad \beta[j] = \sum_{i=1}^{j-1} B[i] \qquad \gamma[j] = \sum_{i=1}^{j-1} \Gamma[i]$$

The runtimes of Algorithms 2 and 3 become $O(|P| \log |P| + d \log |P|)$ and $O(|P| \log(|P|/\varepsilon) + d \log |P|)$ respectively.

The above approach works for any set P. In Algorithm 4, we will use a set P with special structure which allow us to construct the prefix sums faster, in $O(|P| + d \log s)$ time; see the proof of Theorem 5.

Algorithm 4: Data-Dependent Bicriteria Algorithm for Unweighted $\mathsf{MSE}(w,s)$

Input: $w \in \mathbb{R}^d$, $s \in \mathbb{N}$, $\varepsilon > 0$
Output: Quantization set Q

$\mathcal{C} \leftarrow s$ clusters from `s-Center-Clustering`(w,s)
Let p_i^{max} and p_i^{min} be the max and min elements in C_i for all $i \in [s]$
Initialize $P \leftarrow \emptyset$
for $i \in [s]$ **do**
  for $j \in [\sqrt{4d/\varepsilon}]$ **do**
    $P \leftarrow P \cup \{(p_i^{\mathsf{max}} - p_i^{\mathsf{min}}) \cdot \sqrt{\varepsilon/4d} \cdot j\}$
Return Algorithm 3 run on w, P, s, ε with uniform weights

The proof of Theorem 5 closely follows that of Lemma 6.1 in [BBBIMV24]. The key difference is in analyzing the quality of the data-dependent coreset and its implications for the ultimate approximation ratio.

Theorem 5. *There exists an algorithm such that given $w \in \mathbb{R}^d$, target quantization set size $s \in \mathbb{N}$, and parameter $\varepsilon > 0$ returns a quantization set $Q \subset \mathbb{R}$ such that $|Q| = 2s - 2$ and $\mathsf{MSE}(w,Q) \le (1+\varepsilon)\,\mathsf{MSE}(w,s)$. The runtime of the algorithm is $O(d \log s + s\sqrt{d/\varepsilon} \log(d/\varepsilon))$.*

[12] As written, Algorithms 2 and 3 solve the problem for $P = w$ because there always exists an optimal quantization set $Q \subseteq w$.

Proof. We will show the existence of a quantization set $Q \subseteq P$ of size $2s-2$ such that $\mathsf{MSE}(w, Q) \leq (1+\varepsilon) \cdot \mathsf{MSE}(w, s)$.

In particular, consider quantization set Q^* of size s which minimizes the mean-squared error. For any $q \in Q^*$, let $q^{\downarrow} := \min\{p \in P : p \geq q\}$ and $q^{\uparrow} := \max\{p \in P : p \leq q\}$. The solution $Q := \{q^{\uparrow}, q^{\downarrow} : q \in Q\}$ is such that (i) $|Q| = 2s-2$ and (ii) $\mathsf{MSE}(w, Q) \leq (1+\varepsilon)\,\mathsf{MSE}(w, s)$. To see that $|Q| = 2s-2$ note that $w_1, w_d \in Q^*$ but $q^{\downarrow} = q^{\uparrow}$ for both.

To see that $\mathsf{MSE}(w, Q) \leq (1+\varepsilon)\,\mathsf{MSE}(w, s)$, consider some element w_i. There are two cases

1. First, suppose $w_i \in [(w_i^{\downarrow}(Q^*))^{\uparrow}, (w_i^{\uparrow}(Q^*))^{\downarrow}]$. In this case,
$$\begin{aligned}\operatorname*{\mathbf{Var}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w,Q)}[\widehat{\boldsymbol{w}}_i] &= ((w_i^{\uparrow}(Q^*))^{\downarrow} - w_i)(w_i - (w_i^{\downarrow}(Q^*))^{\uparrow}) \\ &\leq (w_i^{\uparrow}(Q^*) - w_i)(w_i - w_i^{\downarrow}(Q^*)) = \operatorname*{\mathbf{Var}}_{\widehat{\boldsymbol{w}} \sim \mathcal{D}^{\mathsf{SSQ}}(w,Q^*)}[\widehat{\boldsymbol{w}}_i]\end{aligned}$$

2. Otherwise, $w_i \in [(w_i^{\downarrow}(Q^*))^{\downarrow}, (w_i^{\downarrow}(Q^*))^{\uparrow}] \cup [(w_i^{\uparrow}(Q^*))^{\downarrow}, (w_i^{\uparrow}(Q^*))^{\uparrow}]$. In this case, w_i is quantized between points from P. The original s-clustering given by the call to `s-Center-Clustering` results in clusters with radius $\leq 4\sqrt{\mathsf{MDV}(w, s)}$. Algorithm 4 then further subdivides and refines the granularity of P to be $\leq 4\sqrt{\mathsf{MDV}(w, s) \cdot \varepsilon/4d}$. Therefore, w_i is quantized in an interval of length $\leq 4\sqrt{\mathsf{MDV}(w, s) \cdot \varepsilon/4d}$ and its variance is then $\leq \mathsf{MDV}(w, s) \cdot \varepsilon/d \leq \mathsf{ADV}(w, s) \cdot \varepsilon/d$.

See Figure 7 for an illustration of the two cases. Taking the sum of variances across all d coordinates then shows that the resulting mean-squared error is at most $(1+\varepsilon)\,\mathsf{MSE}(w, s)$.

Therefore, any solution found by running Algorithm 3 with candidate quantization points P will have objective value $(1+\varepsilon)^2 \cdot \mathsf{MSE}(w, s)$. Thus, running Algorithm 4 with accuracy parameter $\varepsilon' = \varepsilon/3$ results in a $(1+\varepsilon)$-approximation algorithm.

Runtime. Finding the 2-approximate clustering using `s-Center-Clustering` takes $O(d \log s)$ time as per Lemma A.3. Computing the minimum and maximum of each cluster takes $O(d)$ time and constructing the coreset P by uniform subdivision in each cluster takes $O(s\sqrt{d/\varepsilon})$ time.

Finally, note that the prefix sums can be computed in time $O(|P| + d \log s)$ time (as opposed to the $O(|P| \log |P| + d \log |P|)$ bound discussed before) by exploiting the structure of coreset P. In constructing P, we may first sort the s centers returned by `s-Center-Clustering`(w, s), and then uniformly subdivide. Thus, we have constructed P in sorted order in $O(|P| + s \log s) = O(s\sqrt{d/\varepsilon})$ time. To compute the prefix sums, for each $i \in [d]$, we can find the unique $j \in [s\sqrt{4d/\varepsilon}]$ for which $w_i \in [p_j, p_{j+1}]$ by binary searching among the s sorted centers, then performing arithmetic to find the subdivision inside which w_i lies. This thus improved the construction time of the prefix sums to $O(|P| + d \log s)$.

So, Algorithm 3 runs in $O(d \log s + s\sqrt{d/\varepsilon} \log(sd/\varepsilon))$ time. Therefore, the total runtime of Algorithm 4 is $O(d \log s + s\sqrt{d/\varepsilon} \log(d/\varepsilon))$. □

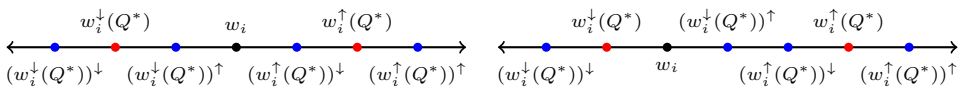


Figure 7: Illustration of (a) case 1 and (b) case 2

We remark that Algorithm 4 is faster than running the exact algorithm on the full instance in regimes where s and ε are modestly valued. As mentioned in Appendix A, however, the use of `s-Center-Clustering` as a subroutine makes Algorithm 4 difficult to implement efficiently in practice.

F Maximum Directional Variance

For any $v \in \mathbb{R}$, we define $\mathsf{MDV}^s(w, v)$ to be the smallest value of s for which $\mathsf{MDV}(w, s) \le v$. We first study MDV^s and give both exact and approximation algorithms. We then use many of these algorithmic primitives and ideas to develop algorithms with small asymptotic runtime and guaranteed $(1+\varepsilon)$-approximations for MDV.

F.1 Exact Algorithms for MDV^s

We first give a simple greedy exact algorithm for MDV^s that runs in time $O(d)$ on sorted vectors.

Algorithm 5: Exact Algorithm for $\mathsf{MDV}^s(w, v)$

Input: Sorted vector $w \in \mathbb{R}^d$ so that $w_1 \le \ldots \le w_d$, variance parameter v
Output: Quantization set Q

```
Initialize Q = {w_1}, q_ℓ = w_1 and q_r = ∞
for i = 2, ..., d − 1 do
    q_r = min{q_r, w_i + v/(w_i − q_ℓ)}
    if q_r ≤ w_i then
        Add q_r to Q
        Set q_ℓ = q_r and update q_r = w_i + v/(w_i − q_ℓ)
if q_r < w_d then
    Add q_r and w_d to Q
else
    Add w_d to Q
return Q
```

Lemma F.1. *If $w \in \mathbb{R}^d$ is sorted, Algorithm 5 solves MDV^s in time $O(d)$.*

Proof. First, we note that any quantization set Q must contain the endpoints w_1 and w_d. Since ASQ requires that every w_i have a quantization point on each side, Q must contain some $q_1 \le w_1$ and $q_{|Q|} \ge w_d$. Replacing q_1 with w_1 and $q_{|Q|}$ with w_d does not increase $\mathsf{VarSSQ}(w_i, Q)$ for any i, so the objective value $\max_{i\in[d]} \mathsf{VarSSQ}(w_i, Q)$ is non-increasing under this substitution.

Second, we prove that the set Q returned by Algorithm 5 has objective value at most v. For each weight w_i, Line 6 computes the furthest right a new quantization point q_r can be placed while maintaining $\mathsf{VarSSQ}(w_j, Q) \le v$ for all j with $w_j < q_r$. Taking the running minimum over these upper bounds ensures the constraint is satisfied for every weight w_i.

Lastly, we prove that any quantization set Q' with $\mathsf{MDV}(w, Q') \le v$ must satisfy $|Q'| \ge |Q|$. Suppose for contradiction that $\mathsf{MDV}(w, Q') \le v$ but $|Q'| < |Q|$. Since Q' is feasible, it must place its second point no further right than q_2, i.e., $q'_2 \le q_2$; otherwise the constraint would be violated for some weight between $q'_1 = w_1$ and q'_2. Applying this argument inductively gives $q'_i \le q_i$ for all $i \le |Q'|$, so the $|Q'| < |Q|$ points of Q' fail to cover all weights in w, contradicting feasibility.

The algorithm makes a single pass over the sorted weights, performing $O(1)$ work per step, so the total runtime is $O(d)$. □

We note that the runtime of Algorithm 5 can be improved to $O(s \log(d/s))$; for all $s \le d$, this is an improvement over the $O(d)$ runtime of Algorithm 5. The algorithm is as follows.

Lemma F.2. *Algorithm 6 outputs a set Q such that $\mathsf{MDV}(w, Q) \le v$ and $|Q| = \mathsf{MDV}^s(w, v)$.*

Proof. The goal of the algorithm is to implement the greedy strategy: given the current largest quantization point q, find the largest possible next point $x^* > q$ such that the variance constraint is satisfied for all $w_j \in [q, x^*]$. Let $x' = 2\sqrt{v} + q$; for all $w_k \in [q, x']$, we have

$$(w_k - q)(x' - w_k) \le \left(\frac{x' - q}{2}\right)^2 = v$$

Algorithm 6: Improved Algorithm for $\mathsf{MDV}^s(w, v)$

Input: Sorted $w \in \mathbb{R}^d$, variance parameter v
Output: Set Q such that $\mathsf{MDV}(w, Q) \le v$ and $|Q| = \mathsf{MDV}^s(w, v)$

```
Initialize Q = {w_1}, the smallest element of w
Let q = w_1 be the max element of Q
while q < w_d do
    Binary search for the smallest index i such that w_i ≥ q + √v
    Set x = min{ w_i + v/(w_i − q), w_{i−1} + v/(w_{i−1} − q) }
    Add x to Q and set q = x
Add w_d to Q
return Q
```

and thus x' is a feasible next quantization point and $x^* \ge x'$. Define $m^* = (x^* + q)/2$ to be the midpoint of $[q, x^*]$; since $x^* \ge x'$, it follows that $m^* \ge q + \sqrt{v}$ as well.

Let w_i be the smallest element of w such that $w_i \ge q + \sqrt{v}$, and consider any y such that $(y+q)/2 > w_i$. Then,

$$(w_i - q)(y - w_i) > (w_i - q)(2w_i - q - w_i) = (w_i - q)^2 \ge v$$

so y is not a feasible next quantization point. Thus, $x^* \in [x', y]$ and $m^* \in [w_{i-1}, w_i]$ as $(x' + q)/2 = q + \sqrt{v} > w_{i-1}$ by construction of i and $(y + q)/2 > w_i$ by definition of y.

Since the function $f(z) = (z - q)(x^* - q)$ is concave and symmetric around $(x^* + q)/2 = m^*$ and $m^* \in [w_{i-1}, w_i]$, the maximum variance of all w_k in $[q, x^*]$ is attained at either w_{i-1} or w_i. Thus, x^* is the largest value for which both w_{i-1} and w_i have variance at most v, which is exactly what the algorithm computes.

The size analysis then follows by the same argument as in the proof of Lemma F.1. □

Lemma F.3. *Let $s = \mathsf{MDV}^s(w, v)$. Then, the above algorithm runs in time $O(s \log(d/s))$ (assuming w is already sorted).*

Proof. Let Q be the set returned by the algorithm, which has size $|Q| \le s$ by Lemma F.2. For each $k \in [s]$, let idx_k denote the index i found in the k-th iteration of the while loop of Algorithm 6.

Since q increases monotonically, the indices idx_k also increase monotonically across iterations. By using the doubling search variant of binary search starting from the previous index idx_{k-1}, the time complexity to find idx_k is $O(\log(\mathsf{idx}_k - \mathsf{idx}_{k-1}))$. So, the total runtime to construct Q is $O\left(\log \prod_{k=1}^{s} B_k\right)$, where $B_k = \mathsf{idx}_k - \mathsf{idx}_{k-1}$. By construction, $\sum_{k=1}^{s} B_k \le d$. Then, by the AM-GM inequality,

$$\prod_{k=1}^{s} B_k \le \left(\frac{\sum_{k=1}^{s} B_k}{s}\right)^s = (d/s)^s$$

and thus the total runtime is $O(\log((d/s)^s)) = O(s \log(d/s))$. □

F.2 Some Basic Approximation Algorithms for MDV^s

In this section, we give algorithms which do not require the $O(d \log d)$-time sorting required for exact algorithms for MDV^s. These algorithms each run in time $O(d \log s)$ and achieve a constant approximation factor.

F.2.1 $(2s, 4v)$-Bicriteria via s-Center Clustering

Algorithm 7: $(2s, 4v)$-Bicriteria Approximation Algorithm

Input: Vector $w \in \mathbb{R}^d$, variance parameter v
Output: quantization set Q

```
Initialize C = ∅
for i ∈ [d] do
    if for all p ∈ C, |w_i − p| > 2√v then
        Add w_i to C
Let Q be the set which contains p − 2√v and p + 2√v for all p ∈ C
return Q
```

Lemma F.4. *Given $w \in \mathbb{R}^d$ and $v \in \mathbb{R}$, let $s = \mathsf{MDV}^s(w, v)$. Then, the above algorithm outputs a set Q of size at most $2s$ such that $\mathsf{MDV}(w, Q) \le 4v$. Moreover, with an appropriate implementation of Line 3, the algorithm runs in time $O(d \log s)$.*

Proof. Since $\mathsf{MDV}(w, s) \le v$ by definition of MDV^s, there exists a set Q^* of size s such that for all i

$$(w_i - w_i^{\downarrow}(Q^*))(w_i^{\uparrow}(Q^*) - w_i) \le v$$

which implies that either $w_i - w_i^{\downarrow}(Q^*) \le \sqrt{v}$ or $w_i^{\uparrow}(Q^*) - w_i \le \sqrt{v}$. Equivalently, every w_i is within $\sqrt{v}$ of some element of Q^* and the set $\{w_i\}_{i \in [d]}$ can be covered with a collection of at most s intervals of length $2\sqrt{v}$.

Let $I_1, \dots, I_s$ be a collection of disjoint intervals of length $2\sqrt{v}$ which cover $\{w_i\}_{i \in [d]}$, and let C be as constructed in Algorithm 7. By construction, for any distinct $x, y \in C$, $|x - y| > 2\sqrt{v}$, and since each I_j has length $2\sqrt{v}$, we thus must have that $|I_j \cap C| \le 1$ for all $j \in [s]$. Moreover, since $\{I_j\}_j$ covers $\{w_i\}_i$ and $C \subseteq \{w_i\}_i$, we have that $|C| = \sum_{j \in [s]} |I_j \cap C| \le s$. Since $|Q| = 2|C|$, we thus have that $|Q| \le 2s$ as desired.

By construction, for all i, there exists a $p \in C$ such that $|w_i - p| \le 2\sqrt{v}$. Equivalently, $w_i \in [p - 2\sqrt{v}, p + 2\sqrt{v}]$. Since both $p - 2\sqrt{v}$ and $p + 2\sqrt{v}$ are in Q, it follows that

$$4\sqrt{v} \ge w_i^{\uparrow}(Q) - w_i^{\downarrow}(Q) = (w_i - w_i^{\downarrow}(Q)) + (w_i^{\uparrow}(Q) - w_i).$$

For any $x, y \ge 0$, $xy \le (x + y)^2/4$, and thus it follows that

$$(w_i - w_i^{\downarrow}(Q))(w_i^{\uparrow}(Q) - w_i) \le 4v$$

as desired. □

F.3 $(1 + \varepsilon)$-approx for MDV^s in time $O(d \log(s/\varepsilon))$

We first need a key structural lemma which can be viewed as a way of constructing a "coreset" for MDV.

Lemma F.5. *Let $I_1, \dots, I_k$ be intervals, each of length at most D, such that $w \subseteq \bigcup_{j \in [k]} I_j$. Let x be the vector of all endpoints of the intervals $I_1, \dots, I_k$, and consider any set $Q \subseteq \mathbb{R}$. Then,*

$$\mathsf{MDV}(w, Q) \le \mathsf{MDV}(x, Q) + \frac{1}{4} \cdot D^2$$

Proof. Let $v = \mathsf{MDV}(x, Q)$, and fix some interval $I_j = [a, b]$. We will show that $\mathsf{VarSSQ}(w_i, Q) \le v + D^2/4$ for all $w_i \in [a, b]$, which then gives the lemma.

First, suppose that $Q \cap [a, b] \neq \emptyset$. Let $q_1 = \min(Q \cap [a, b])$ and $q_2 = \max(Q \cap [a, b])$ (note $q_1 = q_2$ if $|Q \cap [a, b]| = 1$). For all $w_i \in [q_1, q_2]$, there exists some $\alpha \in [0, 1]$ such that $w_i - q_1 \le \alpha D$ and $q_2 - w_i \le (1 - \alpha)D$, as $q_2 - q_1 \le b - a \le D$. Thus,

$$\mathsf{VarSSQ}(w_i, Q) \le \alpha D \cdot (1 - \alpha)D \le \frac{1}{4} \cdot D^2$$

as $\alpha(1 - \alpha) \le 1/4$ for all $\alpha \in [0, 1]$.

So, consider $w_i \in [a, q_1)$. Let $B = q_1 - a$ and $L = a - a^{\downarrow}(Q)$, where $a^{\downarrow}(Q)$ is the largest element of Q which is at most a. By assumption on Q, $BL = \mathsf{VarSSQ}(a, Q) \le v$. For all $w_i \in [a, q_1)$, by definition of B, there exists some $\alpha \in [0, 1)$ such that $w_i = a + \alpha B$. So,

$$\begin{aligned}\mathsf{VarSSQ}(w_i, Q) &= (L + \alpha B) \cdot (1 - \alpha)B \\ &= (1 - \alpha)BL + \alpha(1 - \alpha)B^2 \\ &\le BL + \alpha(1 - \alpha)D^2 \\ &\le v + \frac{1}{4} \cdot D^2\end{aligned}$$

as $BL \le v$, $B \le D$ and $\alpha(1 - \alpha) \le 1/4$ for all $\alpha \in [0, 1]$.

An analogous argument shows that for all $w_i \in (q_2, b]$, $\mathsf{VarSSQ}(w_i, Q) \le v + D^2/4$, thus completing the case where $Q \cap [a, b] \neq \emptyset$.

Now, suppose $Q \cap [a, b] = \emptyset$, and let $L_1 = a - a^{\downarrow}(Q)$, $L_2 = b^{\uparrow}(Q) - b$, where $b^{\uparrow}(Q)$ is the smallest element of Q which is at least b. Let $D_j = b - a$. Then,

$$\begin{aligned}(D_j + L_1)L_2 &= \mathsf{VarSSQ}(b, Q) \le v \\ (D_j + L_2)L_1 &= \mathsf{VarSSQ}(a, Q) \le v\end{aligned}$$

Together, this implies that $D_j L + L_1 L_2 \le v$, where $L = \max(L_1, L_2)$.

Fix some $w_i \in [a, b]$ and set $\alpha \in [0, 1]$ so that $w_i = a + \alpha D_j$ (and thus $w_i = b - (1 - \alpha)D_j$). Then,

$$\begin{aligned}\mathsf{VarSSQ}(w_i, Q) &= (\alpha D_j + L_1) \cdot ((1 - \alpha)D_j + L_2) \\ &= \alpha D_j L_2 + (1 - \alpha)D_j L_1 + L_1 L_2 + \alpha(1 - \alpha)D_j^2 \\ &\le \alpha D_j L + (1 - \alpha)D_j L + L_1 L_2 + \alpha(1 - \alpha)D^2 \\ &= (D_j L + L_1 L_2) + \alpha(1 - \alpha)D^2 \\ &\le v + \frac{1}{4}D^2\end{aligned}$$

as $D_j L + L_1 L_2 \le v$, $D_j \le D$ and $\alpha(1 - \alpha) \le 1/4$. □

Lemma F.5 suggests the following approximation algorithm: construct a small number of intervals, and run an exact algorithm for MDV^s on only the endpoints of those intervals.

Algorithm 8: $(1 + \varepsilon)$-Approximation Algorithm

Input: Vector $w \in \mathbb{R}^d$, variance parameter v, error tolerance ε
Output: Set $Q \subseteq \mathbb{R}$ such that $\mathsf{MDV}(w, Q) \le (1 + \varepsilon)v$ and $\mathsf{MDV}(w, |Q|) \ge v$

Initialize $C = \emptyset$
Iterate through $i \in [d]$ and add w_i to C if for all $c \in C$, $|w_i - c| > 2\sqrt{v}$
for *all* $c \in C$ **do**
  Compute $I_c = \{w_i \mid c = \operatorname{argmin}_{p \in C} |w_i - p|\}$
  Compute $a_c = \min(I_c)$, $b_c = \max(I_c)$, and $D_c = b_c - a_c$
  Compute subintervals $I_c^j = [a_c + D_c(j - 1)\sqrt{\varepsilon}/2,\ a_c + D_c j\sqrt{\varepsilon}/2]$ for each $j \in \lceil 2/\sqrt{\varepsilon} \rceil$
for *each* $c \in C,\ j \in \lceil 2/\sqrt{\varepsilon} \rceil$ **do**
  Compute endpoints $\ell_c^j = \min(w \cap I_c^j)$ and $r_c^j = \max(w \cap I_c^j)$
Let x be a vector of all ℓ_c^j and r_c^j. Run Algorithm 6 on x and v to get a set Q
return Q

Lemma F.6. *Given w, v, ε, Algorithm 8 outputs a set Q such that $\mathsf{MDV}(w, Q) \le (1 + \varepsilon)v$ and $|Q| \le \mathsf{MDV}^s(w, v)$.*

Proof. We first show $|Q| \le \mathsf{MDV}^s(w, v)$. By the analysis of the exact algorithm, we have $|Q| \le \mathsf{MDV}^s(x, v)$. Furthermore, by construction, $x \subseteq w$, and so $|Q| \le \mathsf{MDV}^s(x, v) \le \mathsf{MDV}^s(w, v)$ as desired.

By Lemma F.2, $\mathsf{MDV}(x, Q) \le v$. x consists of the endpoints of the set of intervals $\{I_c^j\}$, which cover w by construction. Moreover, each interval I_c^j has length at most $\sqrt{\varepsilon} D_c/2 \le 2\sqrt{\varepsilon v}$, as $D_c \le 4\sqrt{v}$ by the construction of C.

So applying Lemma F.5, we have that

$$\mathsf{MDV}(w, Q) \le v + \frac{1}{4} \cdot (2\sqrt{\varepsilon v})^2 = (1+\varepsilon)v$$

as desired. □

Lemma F.7. *Given w, v, ε, let $s = \mathsf{MDV}^s(w, v)$. Then, the runtime of Algorithm 8 is*

$$O\left(d \log(s/\varepsilon)\right).$$

Proof. Let C, I_c for each $c \in C$ be defined as in Algorithm 8. By Lemma F.4, $|C| \le s$ and takes time $O(d \log s)$ to construct. Similarly, computing all I_c can be done in time $O(d \log s)$ by iterating through the d points of w and, for each, using binary search to find the closest $c \in C$.

To compute a_c, b_c, we iterate through each of the d elements of w and maintain the min and max elements of I_c encountered so far, for all $c \in C$. Checking which I_c a given w_i lies in takes time $O(\log s)$; updating the stored min/max if needed for I_c can then be done in $O(1)$ time. Similarly, we can compute all $O(s/\sqrt{\varepsilon})$ values ℓ_c^j, r_c^j, and thus the vector x in Line 9, in time $O(d \log s)$ (since the subintervals I_c^j are equally spaced in I_c, for each $y \in I_c$, finding the subinterval in which it lies takes $O(1)$ time).

Finally, sorting and running Algorithm 6 takes time $O(|x| \log(|x|))$. Note that $|x| \le d$, since it is a subset of w, and also $|x| = O(s/\sqrt{\varepsilon})$, by construction. So, $O(|x| \log |x|) = O(|x| \log(s/\sqrt{\varepsilon})) = O(d \log(s/\varepsilon))$.

Thus the total time is $O(d \log(s/\varepsilon))$. □

F.4 Main results for MDV

In this section, we give algorithms for constructing a quantization set of size s with maximum variance approximately $\mathsf{MDV}(w, s)$; in other words, approximation algorithms for $\mathsf{MDV}(w, s)$.

Combined with algorithms from the previous section (i.e. approximation algorithms for MDV^s), we are able to give algorithms with matching runtime of $O(d \log(s/\varepsilon))$.

Theorem 6. *There exists a randomized algorithm that for any given vector $w \in \mathbb{R}^d$, target quantization set size $s \in \mathbb{N}$, and $\varepsilon > 0$, returns a quantization set Q such that $|Q| = s$ and $\mathsf{MDV}(w, Q) \le (1+\varepsilon) \cdot \mathsf{MDV}(w, s)$. The runtime of the algorithm is $O(d \log(s/\varepsilon))$ with probability 0.99.*

F.5 A 4-approximation to MDV

An important subroutine of our algorithm for Theorem 6 is a fast 4-approximation, which we will run on a small coreset $u \in \mathbb{R}^k$, that gives a rough (i.e. $O(1)$-approximate) estimate of the true optimal cost $\mathsf{MDV}(u, s)$. In this subsection, we prove the following lemma.[13].

Lemma F.8. *For any $u \in \mathbb{R}^k$, $s \in \mathbb{N}$ and $\delta \in (0, 1)$, there exists an algorithm which outputs a 4-approximation to the value $\mathsf{MDV}(u, s)$. Moreover, with probability at least $1 - \delta$, the algorithm runs in time*

$$O(k \log k + \sqrt{k} \cdot \log^3 k \cdot \log(1/\delta)).$$

Our first observation is that, in order to obtain a 4-approximation, it suffices to consider quantization sets which only include points from the input vector.

Lemma F.9. *Consider any $u \in \mathbb{R}^k$ and $Q \subset \mathbb{R}$. There exists a $Q' \subset \{u_i\}_{i \in [k]}$ such that $|Q'| = |Q|$ and $\mathsf{MDV}(u, Q') \le 4\,\mathsf{MDV}(u, Q)$.*

[13]The algorithm is described as a Las Vegas algorithm, i.e. the runtime is probabilistic while the quality guarantee is deterministic. This can be easily converted to a Monte Carlo algorithm with deterministic runtime and probabilistic quality by simply terminating the algorithm if it runs for too many steps.

Proof. Reindex so that $u_1 \leq u_2 \leq \ldots \leq u_k$. Construct Q' to be Q, but with each element $q \in Q$ replaced by the nearest element of $\{u_i\}_{i\in[k]}$ to q (breaking ties arbitrarily). By construction, $|Q'| = |Q|$.

Consider some i such that $u_i^{\downarrow}(Q) \notin Q'$, and let $q' = u_j$ be the nearest element of $\{u_i\}_{i\in[k]}$ to $u_i^{\downarrow}(Q)$ (so, $q' \in Q'$). Since, by construction, there are no elements of $\{u_i\}$ in $(u_i^{\downarrow}(Q), q')$, it follows that $q' \leq u_i$.

First, suppose $q' > u_i^{\downarrow}(Q)$. Then, as $u_i \geq q'$, $u_i - u_i^{\downarrow}(Q') \leq u_i - q' < u_i - u_i^{\downarrow}(Q)$.

So, now consider the case where $q' < u_i^{\downarrow}(Q)$, and let $u_j = q'$. Then, $u_i^{\downarrow}(Q)$ is closer to u_j than u_{j+1}, and so

$$(u_i - q') - (u_i - u_i^{\downarrow}(Q)) = |u_i^{\downarrow}(Q) - u_j| \leq |u_i^{\downarrow}(Q) - u_{j+1}| = u_{j+1} - u_i^{\downarrow}(Q)$$

As there are no elements of $\{u_i\}$ in $(u_j, u_{j+1}) = (q', u_{j+1})$ by construction, it follows that $u_i \geq u_{j+1}$. So, $u_{j+1} - u_i^{\downarrow}(Q) \leq u_i - u_i^{\downarrow}(Q)$ and it follows from the above inequality that

$$u_i - u_i^{\downarrow}(Q') \leq u_i - q' \leq (u_i - u_i^{\downarrow}(Q)) + (u_{j+1} - u_i^{\downarrow}(Q)) \leq 2(u_i - u_i^{\downarrow}(Q)).$$

Therefore, for all i, $u_i - u_i^{\downarrow}(Q') \leq 2(u_i - u_i^{\downarrow}(Q))$. An analogous argument shows that for all i, $u_i^{\uparrow}(Q') - u_i \leq 2(u_i^{\uparrow}(Q) - u_i)$, and so for all i,

$$(u_i - u_i^{\downarrow}(Q'))(u_i^{\uparrow}(Q') - u_i) \leq 4(u_i - u_i^{\downarrow}(Q))(u_i^{\uparrow}(Q) - u_i)$$

implying $\mathsf{MDV}(u, Q') \leq 4\,\mathsf{MDV}(u, Q)$. □

F.5.1 Sorted Matrices and Algorithms

Recall from Definition A.1 that a sorted matrix M is such that all rows and columns are sorted in non-decreasing order, i.e. for all i, j, $M[i, j] \geq M[i + 1, j]$ and $M[i, j] \leq M[i, j + 1]$.

Sorted matrices have several useful properties, one of which is that we can efficiently determine all elements of the matrix which lie in a given interval.

Lemma F.10. *Given any sorted matrix M of dimension $k \times k$ and interval $(v_1, v_2) \subset \mathbb{R}$, there exist indices $\ell_1, \ldots, \ell_k$ and $r_1, \ldots, r_k$ such that for all $i \in [k]$, $M[i, j] \in (v_1, v_2)$ if and only if $j \in [\ell_i, r_i]$ (or $\ell_i = r_i = \perp$ if no such elements exist). Moreover, there is an $O(k)$ time algorithm to compute $\ell_1, \ldots, \ell_k$ and $r_1, \ldots, r_k$, which probes $O(k)$ elements of the matrix M.*

Proof. We show how to compute the values ℓ_i; the r_i can be computed analogously by finding the largest element in each row which is strictly less than v_2. The algorithm is as follows[14]:

1. Initialize $j = 1$.
2. For each $i \in [k]$:
   (a) While $j \leq k$ and $M[i, j] \leq v_1$, increment j.
   (b) Set $\ell_i = j$.

Note that the while loop increments j at most k times over the entire algorithm, so the total time is $O(k)$ (and only $O(k)$ elements of M are probed).

For correctness, fix some row i. By construction of the algorithm and the sortedness of the matrix, it follows that $M[i, \ell_i] > v_1$, and thus for all $j \geq \ell_i$, $M[i, j] > v_1$ as well. Suppose for contradiction that there exists some $j < \ell_i$ such that $M[i, j] > v_1$. By construction of the algorithm, we then must have had $M[i', j] \leq v_1$ for some row $i' < i$; however, by sortedness of M, we have $M[i', j] \geq M[i, j] > v_1$, a contradiction.

Finally, for any i where $\ell_i = r_i$ but $M[i, \ell_i] \notin (v_1, v_2)$, set $\ell_i = r_i = \perp$. □

Given such $\ell_1, \ldots, \ell_k$ and $r_1, \ldots, r_k$, we can also efficiently sample a uniformly random element of M which lies in (v_1, v_2).

[14] Note that if no element of M is strictly greater than v_1, then the algorithm will return $\ell_i = k$; this is not a problem, as the final indices will check if $M[i, \ell_i]$ is in (v_1, v_2) and set $\ell_i = r_i = \perp$ if not.

Lemma F.11. *Let M be a sorted matrix of dimension $k \times k$, and let $\ell_1, \ldots, \ell_k$ and $r_1, \ldots, r_k$ be as in Lemma F.10 for some interval (v_1, v_2). With $O(k)$ preprocessing, there is a data structure which can output a uniformly random element of M which lies in (v_1, v_2) in time $O(\log k)$ and probes only 1 element of M.*

Proof. The data structure is as follows. At construction time, first let $S = \{i_1, \ldots, i_m\}$ be the (sorted) list of i such that $\ell_i \neq \perp$; i.e. the rows for which there exists some element of M in (v_1, v_2). For each $j \in [m]$, compute $B_j = (r_{i_j} - \ell_{i_j}) + 1$ and $C_{i_j} = \sum_{h=1}^{j} B_h$. This preprocessing takes $O(k)$ time by first removing the rows for which $\ell_i = r_i = \perp$ and then computing C_{i_j} as prefix sums.

To sample a uniformly random element of M in (v_1, v_2), first sample a uniformly random integer $\boldsymbol{h}$ in $[C_{i_m}]$. Then, use binary search to find the largest element i of S with $C_i \leq \boldsymbol{h}$, and return $M[i, \ell_i + (\boldsymbol{h} - C_i)]$.

Since all elements of M in (v_1, v_2) are exactly those in the intervals $[\ell_i, r_i]$ for $i \in S$, there are C_{i_m} such elements and each is returned with probability $1/C_{i_m}$. The runtime follows from the fact that the binary search takes time $O(\log |S|) = O(\log k)$. □

F.5.2 The algorithm

Definition ($C_u(i,j)$). Given a sorted vector u and $i \leq j$, define $C_u(i,j)$ to be the max variance of u_h for $h \in [i,j]$ when quantized to the endpoints u_i, u_j; that is, $C_u(i,j) = \max_{i \leq h \leq j} (u_h - u_i)(u_j - u_h)$.

Observation F.12. Let C_u be the matrix such that $C_u[i,j] = C_u(i,j)$ for $i \leq j$ and $C_u[i,j] = 0$ otherwise. Then, C_u is a sorted matrix.

Proof. First, note that for any fixed i, $C_u(i,j)$ is non-decreasing in j; similarly, for any fixed j, $C_u(i,j)$ is non-increasing in i. Thus, $C_u[i,j] \leq C_u[i,j+1]$ and $C_u[i,j] \leq C_u[i-1,j]$. □

Observation F.13. When u is sorted, $C_u(i,j)$ can be computed in $O(\log k)$ time.

Proof. Note that $C_u(i,j)$ is maximized at the point u_h closest to the midpoint of $[u_i, u_j]$: $(x - u_i)(u_j - x)$ is increasing for $x \leq (u_i + u_j)/2$ and decreasing for $x \geq (u_i + u_j)/2$, and is symmetric about the midpoint. Thus, we can binary search for the u_h closest to the midpoint $(u_i + u_j)/2$ in time $O(\log k)$, and compute $C_u(i,j)$ in $O(1)$ time given u_h. □

Lemma F.14. *For any sorted $u \in \mathbb{R}^k$ and $Q \subseteq u$, there exists i, j such that $\mathsf{MDV}(u, Q) = C_u(i,j)$.*

Proof. Let $Q = \{q_1 < q_2 < \ldots < q_s\}$, and let β_i be the index such that $q_i = u_{\beta_i}$. Then, by definition,

$$\mathsf{MDV}(u, Q) = \max_{j \in [s-1]} \max_{\beta_j \leq i \leq \beta_{j+1}} (u_i - q_j)(q_{j+1} - u_i) = \max_{j \in [s-1]} C_u(\beta_j, \beta_{j+1}).$$

So, there exists some j such that $\mathsf{MDV}(u, Q) = C_u(\beta_j, \beta_{j+1})$. □

Lemma F.15. *For each $i \in [0, 1, \ldots, \log k/2]$, with probability at least $1 - 2i\delta/\log k$, at the end of the ith iteration of the first `for` loop of Algorithm 9 there are at most $k^2 \cdot (1/2 + 1/\log k)^i$ elements of C_u which lie in the interval (v_1, v_2).*

Proof. We proceed by induction on i. The base case of $i = 0$ is trivial, as C_u contains only k^2 elements. So, fix some $i \geq 1$ and suppose the claim holds for $i - 1$. Let $\boldsymbol{m}$ denote the number of elements of C_u in (v_1, v_2) at the end of the $i - 1$ iteration, so by the inductive hypothesis, with probability at least $1 - 2(i-1)\delta/\log k$, $\boldsymbol{m} \leq k^2(1/2 + 1/\log k)^{i-1}$.

If $\boldsymbol{m} \leq k^2(1/2 + 1/\log k)^i$, then since iteration i can only shrink the interval (v_1, v_2), the count at the end of the iteration is at most $\boldsymbol{m} \leq k^2(1/2 + 1/\log k)^i$ and the claim holds. It therefore suffices to consider the case $\boldsymbol{m} > k^2(1/2 + 1/\log k)^i$.

Algorithm 9: 4-Approximation Algorithm for $\mathsf{MDV}(u, s)$

Input: Sorted $u \in \mathbb{R}^k$, size parameter $2 \le s < k$, failure probability δ
Output: Value v such that $\mathsf{MDV}(u, s) \le v \le 4\,\mathsf{MDV}(u, s)$ with probability at least $1 - \delta$

Initialize $v_1 = 0$, $v_2 = \max_{i,j}\{C_u(i, j)\} = C_u(1, k)$
for $i \in [1, 2, \ldots, \log k/2]$ **do**
  Sample and compute $2^{i+2} \cdot \log^2 k \cdot \log(\log k/\delta)$ elements of C_u uniformly at random
  Let v' be the median of the sampled elements which lie in (v_1, v_2)
  Run Algorithm 6 on u with variance parameter v', which outputs a set of size s'
  if $s' > s$ **then** set $v_1 = v'$;
  else set $v_2 = v'$;
▷ At this step, the number of elements of C_u in (v_1, v_2) is $O(k^{3/2})$ w.p. $1 - \delta$ (Lemma F.15)
Use Lemma F.10 on C_u with interval $I_1^* = (v_1, v_2)$ to compute indices $\ell_1, \ldots, \ell_k$ and $r_1, \ldots, r_k$
for $i \in [1, 2, 3, \ldots, \log k/2]$ **do**
  Use Lemma F.11 to sample and compute $2^{i+4} \cdot \log^2 k \cdot \log(\log k/\delta)$ elements of C_u with values in I_1^*.
  Let v' be the median of the sampled elements which lie in (v_1, v_2)
  Run Algorithm 6 on u with variance parameter v', which outputs a set of size s'
  if $s' > s$ **then** set $v_1 = v'$;
  else set $v_2 = v'$;
▷ At this step, the number of elements of C_u in (v_1, v_2) is $O(k)$ w.p. $1 - \delta$ (Lemma F.17)
Use Lemma F.10 on C_u with interval $I_2^* = (v_1, v_2)$ to compute indices $\ell_1, \ldots, \ell_k$ and $r_1, \ldots, r_k$
Set $\mathcal{S}$ to be the elements of C_u in (v_1, v_2) by computing $C_u[i, j]$ for all $j \in [k]$, $i \in [\ell_j, r_j]$
Add v_2 to $\mathcal{S}$
Sort $\mathcal{S}$, and binary search over $\mathcal{S}$ using Algorithm 6 to find the smallest $v^* \in \mathcal{S}$ such $\mathsf{MDV}^s(u, v^*) \le s$
return v^*

In iteration i, Algorithm 9 draws $T_i = 2^{i+2}L$ entries uniformly at random from all $k(k-1)/2 \le k^2$ entries of C_u, where $L = \log^2 k \log(\log k/\delta)$. Let $\boldsymbol{Y}$ be the random variable of the number of these entries falling in (v_1, v_2). Since $\boldsymbol{m}/k^2 > (1/2 + 1/\log k)^i$,

$$\mathbb{E}[\boldsymbol{Y}] = T_i \cdot \frac{\boldsymbol{m}}{k^2} > 2^{i+2}L \cdot \left(\frac{1}{2} + \frac{1}{\log k}\right)^i = 4\left(1 + \frac{2}{\log k}\right)^i L \ge 4L.$$

By the Chernoff bound,

$$\mathbf{Pr}\left[\boldsymbol{Y} < \mathbb{E}[\boldsymbol{Y}]/4\right] \le \exp\left(-\frac{9\,\mathbb{E}[\boldsymbol{Y}]}{32}\right) \le \exp\left(-\frac{9L}{8}\right) < \frac{\delta}{\log k}.$$

Thus, $\boldsymbol{Y} \ge L$ with probability at least $1 - \delta/\log k$.

Among the $\boldsymbol{Y}$ sampled entries in (v_1, v_2), Algorithm 9 takes their sample median $\boldsymbol{v}'$. Let $\boldsymbol{R}_1$ denote the number of elements of C_u in $(v_1, \boldsymbol{v}')$ and let $\boldsymbol{R}_2$ be the number of elements of C_u in $(\boldsymbol{v}', v_2)$. We will show that, conditioned on $\boldsymbol{Y} \ge L$, with probability at least $1 - \delta/\log k$, both $\boldsymbol{R}_1, \boldsymbol{R}_2$ are at most $(1/2 + 1/\log k)\boldsymbol{m}$.

Let r be the largest entry of C_u with rank less than $(1/2 - 1/\log k)\boldsymbol{m}$. Then, by definition, each of the $\boldsymbol{Y} \ge L$ sampled entries in (v_1, v_2) is less than or equal to r with probability at most $1/2 - 1/\log k$. Since $\boldsymbol{v}'$ is the sample median, it follows that $\boldsymbol{R}_2 \ge (1/2 + 1/\log k)\boldsymbol{m}$ if and only if at least half the sampled elements are less than or equal to r. So, by Hoeffding's inequality,

$$\mathbf{Pr}[\boldsymbol{R}_2 > (1/2 + 1/\log k)\boldsymbol{m} \mid \boldsymbol{Y} \ge L] \le e^{-2L/\log^2 k} \le \frac{\delta}{2\log k}.$$

An analogous argument shows that $\mathbf{Pr}[\boldsymbol{R}_1 > (1/2 + 1/\log k)\boldsymbol{m} \mid \boldsymbol{Y} \ge L] \le \delta/2\log k$. Since $\boldsymbol{Y} \ge L$ with probability at least $1 - \delta/\log k$, a union bound gives

$$\mathbf{Pr}\left[\boldsymbol{R}_1 \le (1/2 + 1/\log k)\boldsymbol{m} \text{ and } \boldsymbol{R}_2 \le (1/2 + 1/\log k)\boldsymbol{m}\right] \ge 1 - 2\delta/\log k$$

At the end of iteration k, the algorithm updates either v_1 or v_2 to $\boldsymbol{v}'$, so the number of points remaining is either $\boldsymbol{R}_1$ or $\boldsymbol{R}_2$. With probability $1 - 2\delta/\log k$, both $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are less than

$(1/2 + 1/\log k)\boldsymbol{m}$, and by the inductive hypothesis, $\boldsymbol{m} \le k^2(1/2 + 1/\log k)^{i-1}$ with probability at least $1 - 2(i-1)\delta/\log k$. So, by the union bound, the number of elements remaining in the interval at the end of iteration k is at most $k^2(1/2+1/\log k)^i$ with probability at least $1 - 2(i-1)\delta/\log k - 2\delta/\log k = 1 - 2i\delta/\log k$. □

Lemma F.15 then immediately gives the following corollary.

Corollary F.16. *At the end of the first* `for` *loop of Algorithm 9, with probability at least* $1 - \delta$*, there are at most* $3k^{3/2}$ *elements of* C_u *in the range* (v_1, v_2).

Proof. The first `for` loop runs for $\log k/2$ iterations. Applying Lemma F.15 at $i = \log k/2$, we have that with probability at least $1 - \delta$, the number of elements of C_u in (v_1, v_2) is at most

$$k^2 \cdot \left(\frac{1}{2} + \frac{1}{\log k}\right)^{\log k/2} = k^2 \cdot \left(\frac{1}{2}\right)^{\log k/2} \cdot \left(1 + \frac{1}{\log k/2}\right)^{\log k/2} \le 3k^{3/2}$$

for sufficiently large k. □

Lemma F.17. *Let* $\mathcal{S}$ *be the set constructed in Line 18 of Algorithm 9. Then, with probability at least* $1 - 2\delta$*,* $|\mathcal{S}| = O(k)$.

Proof. By Corollary F.16, after the end of the first `for` loop, with probability at least $1 - \delta$, there are at most $3k^{3/2}$ elements of C_u which lie in (v_1, v_2). The algorithm of the second `for` loop then samples only from these elements. So, an analogous argument to the proofs of Lemma F.15 and Corollary F.16 but replacing the k^2 total elements of C_u with the $3k^{3/2}$ in (v_1, v_2) gives that there are at most $3k^{3/2} \cdot 3k^{-1/2} = O(k)$ elements of $\mathcal{S}$ with probability at least $1 - \delta$. A union bound then gives the result. □

Lemma F.18. *Algorithm 9 runs in time*

$$O\left(k \log k + \sqrt{k} \cdot \log^3 k \cdot \log(\log k/\delta)\right)$$

with probability at least $1 - 2\delta$.

Proof. Throughout the two `for` loops, there are $\log k$ iterations, each of which requires $T = O(2^{\log k/2} \cdot \log^2 k \cdot \log(\log k/\delta)) = O(\sqrt{k} \cdot \log^2 k \cdot \log(\log k/\delta))$ samples, so the algorithm queries T elements of C. Each element of C can be computed in time $O(\log k)$, so the total time spent computing elements of C is $O(T \log k)$. There are also two calls to the algorithm of Lemma F.10, each of which involves $O(k)$ probes to C and $O(k)$ additional time, so the total time spent on these calls is $O(k \log k)$.

By Lemma F.17, with probability at least $1 - 2\delta$, $|\mathcal{S}| = O(k)$. So, sorting and computing the values of $\mathcal{S}$ takes time $O(k \log k)$. Finally, the binary search of Line 20 takes time $O(k \log k)$, as Algorithm 6 has runtime $O(s \log(k/s)) = O(k)$.

Combining all the steps, the total time is $O(k \log k + T \log k) = O(k \log k + \sqrt{k} \cdot \log^3 k \cdot \log(\log k/\delta))$. □

Lemma F.19. *Algorithm 9 returns a value* v *such that* $\mathsf{MDV}(u, s) \le v \le 4\,\mathsf{MDV}(u, s)$.

Proof. Let v^* be the smallest element of C_u for which $\mathsf{MDV}^s(u, v^*) \le s$. We claim that at all times throughout the execution of the algorithm, $v^* \in (v_1, v_2]$.

Initially, $v_1 = 0 < v^*$[15] and $v_2 = C_u(1, k) \ge v^*$ (since $C_u(1, k)$ is the maximum variance when including only the two endpoints u_1, u_k). At every step of the algorithm, the algorithm updates v_1 to a value v' only if $\mathsf{MDV}^s(u, v') > s$, and thus it must be that $v^* > v'$ as $\mathsf{MDV}^s(u, v^*) \le s$ by construction. Similarly, the algorithm only updates v_2 to a value v' if $\mathsf{MDV}^s(u, v') \le s$. Since the algorithm only ever considers updates to v_2 of the form $v' = C_u(i, j)$ for some $i, j \in [k]$, and v^* is the minimum such value for which $\mathsf{MDV}(u, v') \le s$, it follows that $v^* \le v_2$ as well.

[15] As $s < k$

The algorithm constructs in Lines 18 and 19 the set $\mathcal{S}$ to be all values of C_u in $(v_1, v_2]$ (with the values v_1, v_2 as set in the prior `for` loops). Since $v^* \in (v_1, v_2]$ and there exists i, j such that $v^* = C_u(i, j)$ by construction, it follows that $v^* \in \mathcal{S}$. Moreover, by definition, v^* must be the smallest element of $\mathcal{S}$ for which $\mathsf{MDV}^s(u, v^*) \leq s$, and thus the algorithm returns v^*. Since $\mathsf{MDV}^s(u, v^*) \leq s$, $v^* \geq \mathsf{MDV}(u, s)$. Finally, by Lemma F.9, $v^* \leq 4\,\mathsf{MDV}(u, s)$, completing the proof. □

F.6 Proving Theorem 6

The algorithm of Theorem 6 now follows from combining a fast k-center algorithm, the 4-approximation of Algorithm 9, and a small modification to the $(1+\varepsilon)$-approximation for MDV^s from Algorithm 8. While the algorithm as written returns the *value* of an approximate solution,

Algorithm 10: 4-Approximation Algorithm for $\mathsf{MDV}(w, s)$

Input: $w \in \mathbb{R}^d$, size parameter $2 \leq s < d$, approximation parameter ε, failure probability δ
Output: Value v such that $\mathsf{MDV}(w, s) \leq v \leq (1+\varepsilon)\,\mathsf{MDV}(w, s)$

1 Let C be a 2-approximate s-center solution over w, computed using `s-Center-Clustering` of [FG88]
2 Let I_c be the points of w closest to $c \in C$. Compute $E = \bigcup_{c \in C} \{\min I_c, \max I_c\}$
3 Sort E and run Algorithm 9 on E with size parameter s and error parameter δ to get an estimate v_{apx}
4 Run Algorithm 6 on E with v_{apx} to get a set Q_{apx}, and compute $v' = \mathsf{MDV}(w, Q_{\mathsf{apx}})$
5 **for** *all* $c \in C$ **do**
6 Compute $a_c = \min(I_c)$, $b_c = \max(I_c)$, and $D_c = b_c - a_c$
7 Compute subintervals $I_c^j = [a_c + D_c(j-1)\sqrt{\varepsilon}/4, a_c + D_c j\sqrt{\varepsilon}/4]$ for each $j \leq \lceil 4/\sqrt{\varepsilon} \rceil$
8 Let $E_{\mathsf{sub}} = \bigcup_{c \in C, j \leq \lceil 4/\sqrt{\varepsilon} \rceil} \{\min I_c^j, \max I_c^j\}$
9 Sort E_{sub}
10 Use binary search and Algorithm 6 to find the smallest $i^* \in \mathbb{Z}$ for which $v_i = (1+\varepsilon/2)^i \in [v'/9, v'(1+\varepsilon/2)]$ and $\mathsf{MDV}^s(E_{\mathsf{sub}}, v_i) \leq s$
11 Return $v_{i^*} \cdot (1+\varepsilon/2)$

constructing a set with at most this value is straightforward: instead of returning $v_{i^*}(1+\varepsilon/2)$, use call Algorithm 6 on E_{sub} with v_{i^*} to obtain a set Q' and return Q'.

Lemma F.20. *Algorithm 10 runs in time*

$$O\left(d\log(s/\varepsilon) + \sqrt{s}\cdot\log^3(s)\cdot\log(1/\delta)\right)$$

with probability at least $1-\delta$.

Proof. Computing C and all I_c takes time $O(d\log s)$, from the `s-Center-Clustering` algorithm of [FG88], and thus it takes time $O(d\log s)$ to compute the set E. $|E| \leq 2s$, as $|C| = s$, and so by Lemma F.18, sorting E and running Algorithm 9 takes time $O(s\log s + s^{0.51}\log(1/\delta))$ w.p. $1-\delta$. Computing the set Q_{apx} takes time $O(s)$ using Algorithm 6, and computing $v' = \mathsf{MDV}(w, Q_{\mathsf{apx}})$ takes time $O(d\log s)$ as $|Q_{\mathsf{apx}}| = s$. Thus, the total runtime before the `for` loop is $O(d\log s + s^{0.51}\log(1/\delta))$.

It takes time $O(d\log(s/\varepsilon))$ to compute the subintervals I_c^j, and the same time to compute E_{sub}. Note that $|E_{\mathsf{sub}}| = O(\min(d, s/\sqrt{\varepsilon}))$, as $E_{\mathsf{sub}} \subseteq w$ and each of the $O(s/\sqrt{\varepsilon})$ subintervals contributes at most 2 points to E_{sub}. So, sorting E_{sub} takes time $O(|E_{\mathsf{sub}}|\log|E_{\mathsf{sub}}|) = O(d\log(s/\varepsilon))$.

Finally, the binary search of Line 10 considers $O(\log 1/\varepsilon)$ options v_i, and on each runs Algorithm 6. Each call to Algorithm 6 takes time $O(s\log(|E_{\mathsf{sub}}|/s)) = O(d)$, and so the total runtime of Line 10 is $O(d\log(1/\varepsilon))$.

Combining all the steps, the total runtime is thus

$$O\left(d\log s + s^{0.51}\log(1/\delta) + d\log(s/\varepsilon) + d\log(1/\varepsilon)\right) = O\left(d\log(s/\varepsilon) + s^{0.51}\log(1/\delta)\right). \quad \square$$

Lemma F.21. *Algorithm 10 returns a value v such that* $\mathsf{MDV}(w, s) \leq v \leq (1+\varepsilon)\,\mathsf{MDV}(w, s)$.

Proof. Let $v^* = \mathsf{MDV}(w, s)$, and let $Q^* \subset \mathbb{R}$ be a set with $\mathsf{MDV}(w, Q^*) = v^*$. Since $\mathsf{VarSSQ}(w_i, Q^*) \le v^*$ for all $i \in [d]$, it follows that for each w_i, there exists an element $q \in Q^*$ such that $|q - w_i| \le \sqrt{v^*}$. Thus, the optimal s-center instance has cost at most $\sqrt{v^*}$, and the `s-Center-Clustering` algorithm of [FG88] returns a set C such that for all $i \in [d]$, there exists a $c \in C$ with $|c - w_i| \le 2\sqrt{v^*}$. For each $c \in C$, it then follows that the interval I_c has length at most $4\sqrt{v^*}$.

Let E be constructed as in Line 2 of Algorithm 10, and Q_{apx} as computed in Line 4. E consists of the endpoints of a set of intervals which cover w and which each have length at most $4\sqrt{v^*}$. So, by Lemma F.5, $v' = \mathsf{MDV}(w, Q_{\mathsf{apx}}) \le \mathsf{MDV}(E, Q_{\mathsf{apx}}) + 4v^*$. Moreover, by the guarantees of Algorithm 9 (Lemma F.19), $\mathsf{MDV}(E, Q_{\mathsf{apx}}) \le 4\,\mathsf{MDV}(E, s)$. Thus,

$$v' \le \mathsf{MDV}(E, Q_{\mathsf{apx}}) + 4v^* = 4\,\mathsf{MDV}(E, s) + 4v^* \le 8v^*$$

where the final inequality follows from $E \subseteq w$ and so $\mathsf{MDV}(E, s) \le \mathsf{MDV}(w, s) = v^*$. By definition of v^*, $\mathsf{MDV}(E, Q_{\mathsf{apx}}) \ge v^*$ and we have that

$$v' \in [v^*, 8v^*].$$

Define the intervals I_c^j and set E_{sub} as in Algorithm 10. Each interval I_c has length at most $4\sqrt{v^*}$, and thus, by construction, each I_c^j has length at most $4\sqrt{v^*\varepsilon}/4 = \sqrt{v^*\varepsilon}$. As E_{sub} is the union of the endpoints of the I_c^j, it thus follows that

$$v^* = \mathsf{MDV}(w, s) \le \mathsf{MDV}(E_{\mathsf{sub}}, s) + \varepsilon v^*/4 \quad \text{and so} \quad \mathsf{MDV}(E_{\mathsf{sub}}, s) \ge (1 - \varepsilon/4)v^*.$$

Let $\nu = \mathsf{MDV}(E_{\mathsf{sub}}, s)$; we have $\nu \le v^* \le v'$ (as $E_{\mathsf{sub}} \subseteq w$) and $\nu \ge (1 - \varepsilon/4)v^* > v'/9$. Thus, $\nu \in [v'/9, v']$.

Now, consider the i such that $v_i = (1 + \varepsilon/2)^i$ is in $[v'/9, v']$ and $\nu \le v_i < (1 + \varepsilon/2)\nu$. Since $v_i \ge \nu$, it follows that $\mathsf{MDV}^s(E_{\mathsf{sub}}, v_i) \le s$, and moreover v_i is the smallest such power of $(1 + \varepsilon/2)$ by construction. As $\nu \in [v'/9, v']$, $v_i \in [v'/9, v'(1 + \varepsilon/2)]$ and so the binary search of Line 10 will return v_i.

Finally, the algorithm returns $v_i(1 + \varepsilon/2)$. We have

$$v_i(1 + \varepsilon/2) \ge \nu(1 + \varepsilon/2) \ge v^*(1 - \varepsilon/4)(1 + \varepsilon/2) > v^*$$

as $\nu \ge (1 - \varepsilon/4)v^*$. Moreover, as $\nu \le v^*$,

$$v_i(1 + \varepsilon/2) \le \nu(1 + \varepsilon/2)^2 \le v^*(1 + \varepsilon)$$

giving the desired bounds. □

Theorem 6 then follows from Lemma F.21 and Lemma F.20.

F.7 Irrational Solutions

While we have presented efficient algorithms that achieve high-precision estimates of the optimal quantization set, it turns out that returning an optimal quantization set that minimizes the MDV objective is impossible under standard bit representation (i.e. floating point) of any precision. In particular, here we show a simple example where the vector $w \in \mathbb{Z}^d$, but $\mathsf{MDV}(w, s)$ is irrational (and thus any optimal quantization set contains irrational values as well).

Lemma F.22. *Let* $w = (0, 2, 5, 8, 10)$ *and* $s = 4$. *Then,* $\mathsf{MDV}(w, s) = 8 - 2\sqrt{7}$.

Proof. The quantization set must contain elements 0 and 10 (the min and max of w), so the problem of finding an optimal quantization set reduces to finding the position of the remaining two elements. Let $Q^* = \{0, x, y, 10\}$ be an optimal quantization set, with $x < y$. Let $Q' = \{0, 3, 7, 10\}$; so $\mathsf{MDV}(w, Q') = 4$ and thus $\mathsf{MDV}(w, Q^*) \le 4$. It then follows that $x \le 5$ and $y \ge 5$: any set Q with no elements in $(5, 10)$ or none in $(0, 5)$ has $\mathsf{MDV}(w, Q) > 6$. Similarly, $x \ge 2$ and $y \le 8$: if $x < 2$, the set $Q' = \{0, 2, y, 10\}$ would have $\mathsf{MDV}(w, Q') < \mathsf{MDV}(w, Q^*)$, and analogously for $y > 8$.

When $x \in [2, 5]$ and $y \in [5, 8]$, we have by definition

$$\mathsf{MDV}(w, Q^*) = \max\{2(x - 2), (5 - x)(y - 5), 2(8 - y)\}$$

Solving, we find the minimum occurs when $x = 6 - \sqrt{7}$, $y = 4 + \sqrt{7}$, and $\mathsf{MDV}(w, Q^*) = 8 - 2\sqrt{7}$. □

F.8 Practical Algorithms

While the algorithm presented in this section has excellent theoretical guarantees, its complexity and use of the `s-Center-Clustering` algorithm of [FG88] makes it challenging to use in practice. As such, in this section we detail how we modify our algorithm to better perform on real-world systems, while retaining theoretical guarantees, and provide performance and accuracy evaluations.

The full pseudocode of our algorithm can be found in Algorithm 11. Algorithm 11 is still guaranteed to output a $(1+\varepsilon)$-approximate solution.

Algorithm 11: Practical Algorithm for $\mathsf{MDV}(w, s)$

Input: $w \in \mathbb{R}^d$, size parameter $s \in \mathbb{N}$, number of initial intervals m, approximation parameter $\varepsilon > 0$
Output: Set Q of size s

Let $a = \min(w)$, $b = \max(w)$
Initialize $\mathcal{I}$ as the $m-1$ equal intervals $\left[a + \frac{(b-a)(j-1)}{m-1},\ a + \frac{(b-a)j}{m-1}\right]$ for $j = 1, \ldots, m-1$
for *each* $I \in \mathcal{I}$ *with* $I \cap w \neq \emptyset$ **do**
    Add $\min\{w_i : w_i \in I\}$ and $\max\{w_i : w_i \in I\}$ to x
Sort x; set $\texttt{lo} = 0$ and $\texttt{hi} = (b-a)^2/4$
while $\texttt{hi} - \texttt{lo} > \varepsilon \cdot \texttt{hi}$ **do**
    if *there exists some interval* $I \in \mathcal{I}$ *with* $\max\{w_i : w_i \in I\} - \min\{w_i : w_i \in I\} > \sqrt{\varepsilon \cdot \texttt{hi}}$ **then**
        Set $\mathcal{I}$ to be the $m' = \left\lceil (b-a)/\sqrt{\varepsilon \cdot \texttt{hi}} \right\rceil$ equal-sized intervals of $[a, b]$.
        For each $I \in \mathcal{I}$ with $I \cap w \neq \emptyset$, add $\min\{w_i : w_i \in I\}$ and $\max\{w_i : w_i \in I\}$ to x.
        Sort x.
    Set $\texttt{mid} = (\texttt{lo} + \texttt{hi})/2$
    Run Algorithm 6 on x with variance parameter $\texttt{mid}$; let s' be the size of the returned set
    if $s' > s$ **then** set $\texttt{lo} = \texttt{mid}$;
    else set $\texttt{hi} = \texttt{mid}$;
return *the set* Q *constructed in the last call to Algorithm 6 which updated* $\texttt{hi}$.

Lemma F.23. *Let Q be the set returned by Algorithm 11 when run on vector $w \in \mathbb{R}^d$ and $s \in \mathbb{N}$. Then,*

$$\mathsf{MDV}(w, Q) \leq (1 + O(\varepsilon))\,\mathsf{MDV}(w, s).$$

Proof. We first claim that throughout the entire iteration of the algorithm, $\texttt{lo} < \mathsf{MDV}(w, s) \leq (1+\varepsilon)\texttt{hi}$. Initially, $\texttt{lo} = 0$ and so $\mathsf{MDV}(w, s) > 0$; in any iteration of the `while` loop, `lo` is increased to a value v if and only if $\mathsf{MDV}(x, s) > v$; since $x \subseteq w$, it follows that $\mathsf{MDV}(w, s) \geq \mathsf{MDV}(x, s) > \texttt{lo}$ as well.

For the other direction, by construction $\mathsf{MDV}(w, s) \leq \texttt{hi}$ at the beginning of the algorithm. At any other iteration, `hi` is updated to `mid` if and only if $\mathsf{MDV}(x, s) \leq|$, where x is the endpoint of the intervals $\mathcal{I}$. By construction, each interval $I \in \mathcal{I}$ has length at most $\sqrt{\varepsilon \cdot \texttt{hi}} \leq \sqrt{\varepsilon \cdot 2\texttt{mid}} < 2\sqrt{\varepsilon \cdot \texttt{mid}}$, as $\texttt{mid} = (\texttt{lo} + \texttt{hi})/2 \geq \texttt{hi}/2$. Thus, applying Lemma F.5,

$$\mathsf{MDV}(w, s) \leq \mathsf{MDV}(x, s) + \varepsilon \cdot \texttt{mid} \leq \texttt{mid} + \varepsilon \cdot \texttt{mid} = (1+\varepsilon)\texttt{mid}$$

as we have $\mathsf{MDV}(x, s) \leq \texttt{mid}$. Thus, when updating `hi` to `mid`, we maintain that $\mathsf{MDV}(w, s) \leq (1+\varepsilon)\texttt{hi}$.

At the end of the algorithm, $\texttt{lo} \geq (1-\varepsilon)\texttt{hi}$, due to the termination condition of the `while` loop, and so $(1-\varepsilon)\texttt{hi} \leq \mathsf{MDV}(w, s) \leq (1+\varepsilon)\texttt{hi}$. Moreover, the algorithm returns a set Q such that $\mathsf{MDV}(x, Q) \leq \texttt{hi}$, where x is the endpoint of a collection of intervals of length at most $\sqrt{\varepsilon \cdot \texttt{hi}}$, so applying Lemma F.5 again,

$$\mathsf{MDV}(w, Q) \leq \mathsf{MDV}(x, Q) + \varepsilon \cdot \texttt{hi}/4 \leq (1+\varepsilon)\texttt{hi} \leq (1+\varepsilon)^2\,\mathsf{MDV}(w, s) < (1+2\varepsilon)\,\mathsf{MDV}(w, s)$$

as desired. □

G LLM Quantization Setup

In this section, we describe in detail the LLM experiments presented in Section 4.5.

Quantized weights. We quantize all 96 MLP weight matrices (W_{gate}, W_{up} and W_{down} in each of the 32 layers) of Qwen3.5-4B, and leave all other weights at their original precision. Every method uses $s = 32$ quantization values, so each weight is stored as a 5-bit index. In the per-row setting, each row of each matrix has its own quantization set (stored as 32 fp16 values), for 5.15 bits per weight; in the per-matrix setting, the set is stored per matrix. Quantized weights are decoded back to the original precision before evaluation.

Input distribution for ADV. Each row w of a weight matrix is used only through inner products $\langle w, x\rangle$ with the input x to that layer, so we take $\mathcal{X}$ to be the distribution of these inputs. We estimate $\lambda_i = \mathbf{E}_{\boldsymbol{x}\sim\mathcal{X}}[\boldsymbol{x}_i^2]$ for each input coordinate i by running the unquantized model on 16 windows of 2048 tokens from the WikiText-2 training set. In the per-row setting, entry w_i of a row has weight λ_i; in the per-matrix setting, entry W_{ji} has weight λ_i. We use the λ_i as real-valued weights in the exact algorithm, rescaling them to have mean 1 (with a minimum weight of 10^{-6}).

Methods. We compare ADV against four baselines. ASQ optimizes the same objective with every coordinate weighted equally, effectively *ignoring* the input ditribution $\mathcal{X}$. k-means gives the quantization set with minimal MSE under deterministic nearest rounding, while uniform quantization is the simplest non-adaptive choice. TurboQuant also targets inner product preservation, but is non-adaptive: it applies a random rotation so that the entries of the weight matrix are approximately Gaussian, and then uses a fixed global quantization set designed for the Gaussian distribution. In detail, the methods are as follows.

- ADV: the optimal quantization set for ADV, computed with our exact algorithm.
- ASQ: the optimal quantization set for (unweighted) ASQ, computed with the same algorithm with all weights equal to 1.
- k-means: the optimal 1D k-means centers, computed with the `Wilber` algorithm of [GLM+17].
- TurboQuant [ZDHM25]: the weight matrix W is replaced by $WR^\top$, where $R = \frac{1}{\sqrt{b}}BDP$ for a random permutation P, a random ± 1 diagonal matrix D, and a block-diagonal matrix B of $b \times b$ Hadamard matrices, with b the largest power of 2 dividing the input dimension. Since R is orthogonal, $Wx = (WR^\top)(Rx)$, so the layer is evaluated on the rotated input Rx. The quantization set is the s-point Lloyd-Max quantizer for $N(0, 1)$, shifted and scaled by the mean and standard deviation of the (rotated) row or matrix.
- Uniform: s evenly spaced values between the minimum and maximum entry.

Although ADV and ASQ are defined in terms of standard stochastic quantization, in Tables 1 and 2 we round each weight to its nearest quantization value for every method. In the per-row setting, quantization sets are computed from all entries of the row, and in the per-matrix setting, from all entries of the matrix.

Since the model's weights are stored in bfloat16 and thus contain many repeated values, before computing the quantization sets for ADV, ASQ and k-means we merge all entries with equal values into a single point whose weight is the sum of the weights of the merged entries. This does not change the objective to be optimized.

Evaluation. We split the WikiText-2 test set into 145 non-overlapping windows of 2048 tokens. At each position, let p and q be the next-token distributions of the unquantized and quantized models. Storing p in full over the 152k-token vocabulary is infeasible, so we keep only the 512 most likely tokens S under p (on average 98.25% of its mass) and report

$$\sum_{t\in S} p_t \log \frac{p_t}{q_t} + p_{\bar{S}} \log \frac{p_{\bar{S}}}{q_{\bar{S}}},$$

where $p_{\bar{S}}$ and $q_{\bar{S}}$ are the total probabilities outside of S. By the data processing inequality, this is a lower bound on $\mathrm{KL}(p \,\|\, q)$. We average over all positions. Top-1 agreement is the fraction of positions

at which both models assign the highest probability to the same token, and perplexity is computed as usual (the unquantized model has perplexity 9.58).

For ARC-Challenge, we use all 1172 questions of the test set. Each question is given to the model as "`Question: <question>\nAnswer:`", each candidate answer is scored by its log-likelihood divided by its length in characters, and the highest-scoring answer is chosen. ARC agree is the fraction of questions on which the quantized model chooses the same answer as the unquantized model, whether or not that answer is correct.

H Deferred Proofs and Figures

H.1 Deferred Proofs from Appendix D

Proof of Lemma D.1. First, note that Adaptive Stochastic Quantization distribution is only well-defined if both $w_1^\downarrow(Q)$ and $w_1^\uparrow(Q)$ exist. We show that if $\{w_1^\downarrow(Q), w_1^\uparrow(Q)\} \neq \{w_1, w_d\}$, then $\mathsf{ADV}_{\mathcal{X}}(w, Q) > \mathsf{ADV}_{\mathcal{X}}(w, s)$. First, suppose that $w_1^\downarrow(Q) \neq w_1$. Then, consider quantization set $Q' := (Q \setminus \{w_1^\downarrow(Q)\}) \cup \{w_1\}$ and observe that

$$\begin{aligned}
\mathsf{ADV}_{\mathcal{X}}(w, Q') &= \sum_{i=1}^{d} \lambda_i (w_i^\uparrow(Q') - w_i)(w_i - w_i^\downarrow(Q')) \\
&= \sum_{i=2}^{d} \lambda_i (w_i^\uparrow(Q') - w_i)(w_i - w_i^\downarrow(Q')) && (w_1^\downarrow(Q') = w_1) \\
&\leq \sum_{i=2}^{d} \lambda_i (w_i^\uparrow(Q) - w_i)(w_i - w_i^\downarrow(Q)) \\
&\leq \mathsf{ADV}_{\mathcal{X}}(w, Q)
\end{aligned}$$

Where the first inequality follows because $w_i^\uparrow(Q) = w_i^\uparrow(Q')$ and $w_i^\downarrow(Q') \geq w_i^\downarrow(Q)$. A symmetric argument can be used to show that $\mathsf{ADV}_{\mathcal{X}}(w, Q) > \mathsf{ADV}_{\mathcal{X}}(w, s)$ if $w_1^\downarrow(Q) \neq w_d$. Thus, any quantization set not containing w_1 and w_d must be suboptimal with respect to $\mathsf{ADV}_{\mathcal{X}}$. □

Proof of Lemma D.2. Consider an optimal quantization set $Q = \{q_1, \ldots, q_s\} \not\subseteq w$ such that $q_1 \leq \ldots \leq q_s$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathsf{ADV}_{\mathcal{X}}(w, s)$. The argument proceeds by iteratively transforming Q, while leaving its objective value unchanged.

Consider $q_i \in Q$ such that $q_i \notin w$. Define $q_i^\downarrow := \max\{w_j : w_j \leq q_i\}$ and analogously $q_i^\uparrow := \min\{w_j : w_j \geq q_i\}$. We first show that $q_{i-1} < q_i^\downarrow$ and $q_{i+1} > q_i^\uparrow$. Suppose that $q_{i-1} \geq q_i^\downarrow$ and $q_{i+1} > q_i^\uparrow$. Then, quantization set $Q' := (Q \setminus \{q_i\}) \cup \{q_i^\uparrow\}$ has $\mathsf{ADV}_{\mathcal{X}}(w, Q') < \mathsf{ADV}_{\mathcal{X}}(w, Q)$, contradicting the optimality of Q. A symmetric argument contradicts the optimality of Q when $q_{i-1} < q_i^\downarrow$ and $q_{i+1} \leq q_i^\uparrow$. When both $q_{i-1} \geq q_i^\downarrow$ and $q_{i+1} \leq q_i^\uparrow$, the quantization point q_i is never rounded to by ASQ on w, therefore quantization set $Q' := (Q \setminus \{q_i\}) \cup \{w_{i'}\}$, where $w_{i'} \notin Q$, has $\mathsf{ADV}_{\mathcal{X}}(w, Q') < \mathsf{ADV}_{\mathcal{X}}(w, Q)$. Thus, for the remainder of the proof, we assume that $q_{i-1} < q_i^\downarrow$ and $q_{i+1} > q_i^\uparrow$.

We construct quantization set $Q_1 := (Q \setminus \{q_i\}) \cup \{q_i^\downarrow\}$. Then,

$$\begin{aligned}
\mathsf{ADV}_{\mathcal{X}}(w, Q_1) - \mathsf{ADV}_{\mathcal{X}}(w, Q) &= \sum_{w_j \in [q_{i-1}, q_i^\downarrow]} \lambda_j (w_j - q_{i-1}) \cdot \left(q_i^\downarrow - q_i\right) \\
&\quad + \sum_{w_k \in [q_i^\downarrow, q_{i+1}]} \lambda_j (q_{i+1} - w_k) \cdot \left(q_i - q_i^\downarrow\right) \\
&= (q_i - q_i^\downarrow) \cdot \left[\sum_{w_k \in [q_i^\downarrow, q_{i+1}]} \lambda_j (q_{i+1} - w_k) - \sum_{w_j \in [q_{i-1}, q_i^\downarrow]} \lambda_j (w_j - q_{i-1}) \right] \\
&= (q_i - q_i^\downarrow) \cdot \gamma
\end{aligned}$$

Analogously, we construct $Q_2 := (Q \setminus \{q_i\}) \cup \{q_i^\uparrow\}$. Then,

$$\begin{aligned}
\mathsf{ADV}_{\mathcal{X}}(w, Q_2) - \mathsf{ADV}_{\mathcal{X}}(w, Q) &= \sum_{w_j \in [q_{i-1}, q_i^\uparrow]} \lambda_j (w_j - q_{i-1}) \cdot \left(q_i^\uparrow - q_i\right) \\
&\quad + \sum_{w_k \in [q_i^\uparrow, q_{i+1}]} \lambda_j (q_{i+1} - w_k) \cdot \left(q_i - q_i^\uparrow\right) \\
&= (q_i^\uparrow - q_i) \cdot \left[\sum_{w_j \in [q_{i-1}, q_i^\uparrow]} \lambda_j (w_j - q_{i-1}) - \sum_{w_k \in [q_i^\uparrow, q_{i+1}]} \lambda_j (q_{i+1} - w_k) \right] \\
&= -(q_i^\uparrow - q_i) \cdot \gamma
\end{aligned}$$

Because we assume that $\mathsf{ADV}_{\mathcal{X}}(w, Q) = \mathsf{ADV}_{\mathcal{X}}(w, s)$, it must be the case that both $\mathsf{ADV}_{\mathcal{X}}(w, Q_1) - \mathsf{ADV}_{\mathcal{X}}(w, Q) \geq 0$ and $\mathsf{ADV}_{\mathcal{X}}(w, Q_2) - \mathsf{ADV}_{\mathcal{X}}(w, Q) \geq 0$. But $(q_i - q_i^\downarrow) > 0$ and $(q_i^\uparrow - q_i) > 0$, so it must be the case that $\gamma = 0$. Therefore, $\mathsf{ADV}_{\mathcal{X}}(w, Q_1) = \mathsf{ADV}_{\mathcal{X}}(w, Q_2) = \mathsf{ADV}_{\mathcal{X}}(w, Q)$. Iteratively replacing each $q_i \notin w$ from Q in this way then proves the claim. □

Proof of Lemma D.3. We prove that C satisfies the Quadrangle Inequality (Definition A.5) Note that for $i \in [a, b]$, we have

$$(w_c - w_i)(w_i - w_a) \leq (w_f - w_i)(w_i - w_a) \tag{1}$$

For $i \in [c, f]$, we have

$$(w_f - w_i)(w_i - w_b) \leq (w_f - w_i)(w_i - w_a) \tag{2}$$

For $i \in [b, c]$, we have

$$\begin{aligned}
&(w_c - w_i)(w_i - w_a) + (w_f - w_i)(w_i - w_b) \\
&\quad = (w_c - w_i)(w_i - w_b) + (w_f - w_i)(w_i - w_a) + (w_a - w_b)(w_f - w_c) \\
&\quad \leq (w_c - w_i)(w_i - w_b) + (w_f - w_i)(w_i - w_a)
\end{aligned} \tag{3}$$

Thus, we have that

$$\begin{aligned}
C[a, c] + C[b, f] &= \sum_{i \in [a,c]} \lambda_i (w_c - w_i)(w_i - w_a) + \sum_{i \in [b,d]} \lambda_i (w_f - w_i)(w_i - w_b) \\
&= \sum_{i \in [a,b]} \lambda_i (w_c - w_i)(w_i - w_a) + \sum_{i \in [c,d]} \lambda_i (w_f - w_i)(w_i - w_b) \\
&\qquad + \sum_{i \in [b,c]} \lambda_i [(w_c - w_i)(w_i - w_b) + (w_f - w_i)(w_i - w_a)] \\
&\leq \sum_{i \in [a,b]} \lambda_i (w_f - w_i)(w_i - w_a) + \sum_{i \in [c,d]} \lambda_i (w_f - w_i)(w_i - w_a) \\
&\qquad + \sum_{i \in [b,c]} \lambda_i \left[(w_c - w_i)(w_i - w_b) + (w_f - w_i)(w_i - w_a)\right] \\
&= \sum_{i \in [a,d]} \lambda_i (w_f - w_i)(w_i - w_a) + \sum_{i \in [b,c]} \lambda_i (w_c - w_i)(w_i - w_b) = C[a, d] + C[b, c].
\end{aligned}$$

Proof of Lemma D.4. We first show that the rows of C are sorted. Consider a fixed row $i \in [d]$ and let $j, k \in [d]$ such that $i < j < k$. Then,

$$C[i, j] = \sum_{\ell=i}^{j} \lambda_\ell (w_j - w_\ell)(w_\ell - w_i) \leq \sum_{\ell=i}^{k} \lambda_\ell (w_k - w_\ell)(w_\ell - w_i) = C[i, k]$$

Similarly, for a fixed column $j \in [d]$ and rows $i < k$ we have

$$C[i, j] = \sum_{\ell=i}^{j} \lambda_\ell (w_j - w_\ell)(w_\ell - w_i) \geq \sum_{\ell=k}^{j} \lambda_\ell (w_j - w_\ell)(w_\ell - w_k) = C[k, j]$$

This proves that matrix C has sorted rows and columns. □

H.2 Deferred Table from Section 2

Table 4: Recall@100 of maximum inner product and ℓ_2 search on vectors from GloVe 300D [CBM25]. The dataset consists of 900,000 randomly selected GloVe vectors, each query is sampled from 100,000 (disjoint from dataset) vectors, and each method is tested on 10,000 sampled queries. We benchmark against the FAISS implementation of Product Quantization (PQ) [MUJS18]; each method uses an average of 4-bits per coordinate.

Task	Method	Average	Worst 1%	Worst 0.1%	Worst
Max-IP	PQ	0.8535	0.7600	0.6600	0.4500
	ADV	0.8751	0.8000	0.7700	0.7400
ℓ_2	PQ	0.8927	0.7800	0.7200	0.6600
	ADV	0.8907	0.7700	0.7200	0.6400

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We provide theoretical and empirical evidence for every one of our claims throughout the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Section 5 for a discussion of the limitations of our work. In all empirical evaluations, we clearly specify the type/distribution of inputs used.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.

- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that are not acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The full proofs of every theorem/lemma statement are provided in the appendices. We provide proof sketches for intuition in the main body.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: In Section 4.1 we provide full details of the hardware on which our runtime experiments are conducted, and in Section 4.5 we fully detail the setup of our LLM quantization experiments. We provide the full details of all our empirical algorithms, the parameters with which they are run, and the data distributions/datasets on which they are run.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often

one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The full code repository is available at `https://github.com/nathanllww/Inner-Product-Aware-Quantization`

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify all distributions used in our testing suite, algorithmic parameters, LLMs, and datasets. See plot captions throughout the paper.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All plots throughout the paper include error ranges indicating the 10-90% result quantiles.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Section 4.1 for a full specification of the hardware used to run all our runtime experiments. For our LLM quantization experiments, the accuracy results do not depend on the hardware used, and the full experimental setup can be found in Section 4.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that did not make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: Our results do not involve human subjects nor the potential for societal harm.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: Our work addresses a foundational primitive found in many systems. It does not address a particular application.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: We do not release any data or models.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use and acknowledge the code and data of [GLM+17] and [CBM25] respectively. Our use of Qwen3.5-4B is allowed under the Apache 2.0 license of the weights.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not release any assets with this submission.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our work does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our work does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We used LLM tools to improve the presentation of this work, and to improve the proof of Theorem 4.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.