FAST-MWEM: PRIVATE DATA RELEASE IN SUBLINEAR TIME

THEMISTOKLIS HARIS, STEVE CHOI, AND MUTIRAJ LAKSANAWISIT

Department of Computer Science, Boston University

tharis@bu.edu, stevec@bu.edu, mutiraj@bu.edu

Abstract. The Multiplicative Weights Exponential Mechanism (MWEM) is a fundamental iterative framework for private data analysis, with broad applications such as answering m linear queries, or privately solving systems of m linear constraints. However, a critical bottleneck hindering its scalability is the $\Theta(m)$ time complexity required to execute the exponential mechanism in each iteration. We introduce a modification to the MWEM framework that improves the per-iteration runtime dependency to $\Theta(\sqrt{m})$ in expectation. This is done via a lazy sampling approach to the Report-Noisy-Max mechanism, which we implement efficiently using Gumbel noise and a k-Nearest Neighbor data structure. This allows for the rapid selection of the approximate score in the exponential mechanism without an exhaustive linear scan. We apply our accelerated framework to the problems of private linear query release and solving Linear Programs (LPs) under neighboring constraint conditions and low-sensitivity assumptions. Experimental evaluation confirms that our method provides a substantial runtime improvement over classic MWEM.

1. Introduction

Differential Privacy (DP) has emerged as the gold standard for private data analysis, providing a rigorous mathematical framework for preventing the leakage of personal information [Dwork et al., 2006]. The core principle of DP is that the outcome of a randomized algorithm should not significantly change if a single individual's data is added to or removed from a dataset. Since its introduction, this definition has led to the development of a wide variety of differentially private algorithms that are often deployed in industry and government [Dwork et al., 2019]. Yet a persistent tension in this literature is that the strongest privacy guarantees often come at steep computational cost — a barrier that limits practical deployment on large datasets.

The **Multiplicative Weights Exponential Mechanism (MWEM)** exemplifies this tension. MWEM combines the Multiplicative Weights Update (MWU) method [Hardt et al., 2012] with the private exponential mechanism [McSherry and Talwar, 2007]: at each iteration, MWU refines a synthetic data distribution, while the exponential mechanism privately selects the query or constraint with the highest error against that distribution. This powerful combination underlies algorithms for private linear query answering [Hardt et al., 2012], linear programming [Hsu et al., 2014], boosting [Bun et al., 2020], and synthetic data generation. Despite its versatility and strong theoretical guarantees, however, MWEM has a fundamental computational bottleneck: the exponential mechanism must evaluate a score for *every* candidate at each iteration, giving a per-iteration runtime that scales *linearly* in the number of queries or constraints m. For large-scale problems this linear dependency is a serious barrier.

Our work challenges this barrier. We show that the per-iteration cost of MWEM's exponential mechanism can be reduced from $\Theta(m)$ to $\widetilde{\Theta}(\sqrt{m})$ in expectation, while *provably* preserving MWEM's privacy and utility guarantees. The key insight is that scores in MWEM are inner products — a

structure that enables a surprising and non-trivial connection to sublinear-time data structures from the nearest-neighbor search literature.

1.1. **Our Results and Techniques.** Our primary contribution is **Fast-MWEM**, an algorithmic framework that reduces the per-iteration time complexity of MWEM's exponential mechanism from $\Theta(m)$ to an expected $\widetilde{\Theta}(\sqrt{m})$, with *provably preserved* (ε, δ)-DP and utility guarantees. We apply Fast-MWEM to two fundamental problems:

Private Linear Query Release.. We first address the classic problem of answering a collection Q of m linear queries on a dataset X consisting of n elements from a domain set $\mathcal{X}$. Viewing the dataset as a normalized histogram vector $h \in [0,1]^{|\mathcal{X}|}$ and each query as a vector $q \in [0,1]^{|\mathcal{X}|}$, our goal is to output a "synthetic histogram" vector $\hat{p}$ that minimizes the maximum error:

$$\max_{q \in Q} |\langle q, h - \hat{p} \rangle| . \tag{1}$$

We provide an algorithm that matches the utility and privacy guarantees of the classic MWEM algorithm but whose expected per-iteration runtime scales with $\tilde{O}(|\mathcal{X}|\sqrt{m})$, compared to the previous $O(|\mathcal{X}|m)$. For settings with large query sets this yields a substantial speedup.

Private Linear Programming.. We also apply our framework to the problem of privately solving Linear Programs (LPs) with m constraints. We focus on LPs of the form

$$\max_{x \in \mathbb{R}^d} \{c^\top x\}, \text{ subject to } Ax \leq b, \tag{2}$$

with $c \in \mathbb{R}^d, A \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^m$. We consider two settings [Hsu et al., 2014] for neighboring databases $D \sim D'$:

- *scalar-private, low-sensitivity* LPs (Section 4): we have $A(D) = A(D')$, $c(D) = c(D')$ and $||b(D) - b(D')||_\infty \leq \Delta_\infty$.
- *constraint-private* LPs (Appendix B.1): the constraints $(A(D), b(D))$ and $(A(D'), b(D'))$ differ in that one of them has one extra row.

Fast-MWEM maintains the utility guarantees of Hsu et al. [2014] while reducing expected per-iteration runtime from $O(dm)$ to $O(d\sqrt{m})$ for scalar-private LPs, and by $O(m\sqrt{d})$ for large-width constraint-private LPs.

Our Techniques. The central challenge is that the exponential mechanism is inherently *global*: its privacy guarantee arises from normalizing over all m candidates, so any truncated scheme must prove that examining fewer candidates does not leak additional information about the private data. This is non-trivial — the selection of *which* candidates to examine could itself be privacy-sensitive.

Our key structural observation is that in MWEM, scores take the form of inner products between a query vector that evolves with the private data and a fixed set of candidates. This unlocks a two-part approach. First, we show that *lazy Gumbel sampling* [Mussmann et al., 2017] — a technique originally designed for fast softmax sampling — can be applied to MWEM's exponential mechanism *without any loss in its privacy guarantee*. The non-trivial step is proving that, conditioned on correctly identifying the top-$\sqrt{m}$ scores, the output distribution of the truncated sampler is *exactly identical* to that of the full exponential mechanism (Theorem 3.3). Second, finding these top-$\sqrt{m}$ scores reduces to k-Maximum Inner Product Search (k-MIPS), for which sublinear-time approximate solvers (LSH [Datar et al., 2004], IVF [Baranchuk et al., 2018], HNSW [Malkov and Yashunin, 2018]) are available.

A further technical contribution is our *approximation analysis* (Theorems G.2 and G.10): since practical MIPS solvers are approximate, we give a formal characterization of what approximate retrieval costs in terms of differential privacy. We prove two complementary tradeoffs — one can absorb the approximation error as a small additive privacy slack while retaining $\Theta(\sqrt{m})$ runtime, or

eliminate the slack entirely at a bounded runtime overhead — providing rigorous guarantees for the approximate solvers used in practice.

Experimental Evaluation. We validate our framework on synthetic benchmarks for both linear query release and private LP solving, and on the real-world ADULT dataset for marginal query release. Our experiments confirm that Fast-MWEM achieves up to 42× speedup in query selection over standard MWEM, with the gap widening as m grows — consistent with our $\Theta(\sqrt{m})$ prediction. Crucially, this acceleration comes with negligible impact on mean error, and the modest increase in worst-case error is precisely accounted for by our approximation theorems.

1.2. **Related Work.** Since its inception [Hardt et al., 2012], MWEM and its variants has been used heavily in private data analysis. In the problem of linear query release, numerous works have provided close to optimal error guarantees while also achieving practical performance [Liu et al., 2021, Roth and Roughgarden, 2010, El Emam et al., 2020]. For private linear programming and optimization, MWEM is a crucial ingredient in many algorithms [Hsu et al., 2014], though recent developments have revealed more efficient algorithms [Ene et al., 2025, Kaplan et al., 2025].

Relation to workload-aware methods. A separate line of work — including HDMM [McKenna et al., 2021], AIM [McKenna et al., 2022], and RAP [Aydore et al., 2021] — pursues a complementary goal: designing query strategies or synthetic-data objectives that reduce error on a given workload, rather than accelerating the selection step. These methods are largely orthogonal to Fast-MWEM. Our contribution is a *computational* speedup to the exponential mechanism that applies whenever MWEM (or a MWEM-like algorithm) is used as a subroutine; it does not address workload-specific error optimization. Combining Fast-MWEM with workload-aware strategies is a natural avenue for future work.

The runtime bottlenecks of MWEM are also inherent to the private linear query release problem. Lower bounds [Ullman and Vadhan, 2011] exist showing that no algorithm can answer more than $n^{2+o(1)}$ queries with constant accuracy in time polynomial in *both* the dimension of the data *and* n. Our results improve the dependency on the number of queries, but maintain an exponential dependence on the data dimension. Combining our approach with heuristics designed for ameliorating this curse of dimensionality is an interesting future direction.

2. Preliminaries

We give the formal definition of differential privacy below:

Definition 2.1. Let $\mathcal{A}$ be any randomized algorithm that operates on databases whose elements come from some universe. For parameters $\varepsilon > 0$ and $\delta \in [0, 1]$, the algorithm $\mathcal{A}$ is (ε, δ)–**differentially private (DP)** if for any two neighboring databases $S \sim S'$ (ones that differ on one row only) and for any subset of outcomes T of the output space, we have:

$$\Pr[\mathcal{A}(S) \in T] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(S') \in T] + \delta.$$

Definition 2.2. The **exponential mechanism (EM)** [McSherry and Talwar, 2007] solves the *private selection* problem: Given a set of candidate options R and a *score function* $s : \mathcal{D} \times R \to \mathbb{R}$, we seek to select the candidate with close-to-maximal score. Let the *global sensitivity* of s be defined as:

$$\Delta := \max_{h \in R} \max_{D \sim D'} \left| s(D, h) - s(D', h) \right|. \tag{3}$$

Given a privacy parameter ε, EM samples a candidate according to a categorical distribution:

$$\Pr[h \text{ is selected} \mid D] \propto \exp\left(\frac{\varepsilon \cdot s(D, h)}{2\Delta}\right). \tag{4}$$

EM has the following formal guarantees:

Theorem 2.3. *The Exponential Mechanism is ε-DP and runs in $O(|R|)$ time. Utility-wise, if EM outputs some candidate $\widehat{h} \in R$, then:*

$$\Pr_{\widehat{h}}\left[s(D,\widehat{h}) < q_{\max} - \frac{2\Delta(\ln|R| + t)}{\varepsilon}\right] \leq e^{-t} \tag{5}$$

where $q_{\max} = \max_{i \in R} s(D, h_i)$ is the maximum score of any candidate.

3. Method & Framework

In this section we present our framework for accelerating MWEM. For ease of presentation, we first focus on the problem of answering linear queries privately.

3.1. **Setting.** Let $\mathcal{X}$ be a domain set and $X = \{x_1, ..., x_n\} \subseteq \mathcal{X}^n$ be a dataset. The goal is to answer a collection of *linear queries*, which are functions $\phi : \mathcal{X} \to [0,1]$. Let us define

$$\phi(X) := \frac{1}{n}\sum_{i=1}^{n} \phi(x_i).$$

The *histogram representation* $h \in [0,1]^{|\mathcal{X}|}$ of dataset X is defined as:

$$h_x = \frac{|\{i \in [n] : x_i = x\}|}{n}, \text{ for all } x \in \mathcal{X}.$$

We can view a linear query as a vector in $[0,1]^{|\mathcal{X}|}$. Then we have that:

$$\phi(X) = \langle \phi, h \rangle.$$

Then, if we are given m queries $q_1, ..., q_m \in [0,1]^{|\mathcal{X}|}$, the answers to all of them can be captured by the product $Q \cdot h$, where $Q = \{q_1, \ldots, q_m\} \in [0,1]^{m \times |\mathcal{X}|}$.

The goal of *Private Linear Query Release* is to design a randomized algorithm $\mathcal{M}$ that satisfies (ε, δ)-differential privacy. Given privacy parameters $\varepsilon, \delta > 0$ and an error bound $\alpha > 0$, the algorithm should output a synthetic distribution vector $\hat{p} \in [0,1]^{|\mathcal{X}|}$ such that, with high probability, the maximum error over all queries is bounded by α:

$$\|Q\hat{p} - Qh\|_\infty = \max_{i \in [m]} |\langle q_i, \hat{p} \rangle - \langle q_i, h \rangle| \leq \alpha.$$

3.2. **The MWEM algorithm.** As a warm-up, let us review the classic MWEM algorithm for solving the private linear query release problem. Intuitively, the iterative MWU part acts as an "analyst" and continuously refines a synthetic data distribution $\widehat{p}$ that achieves low error. The exponential mechanism acts as private "adversary", finding queries with high error against the chosen synthetic data distribution.

We present MWEM as Algorithm 1 in two parts: the MWU *iterative part*, and the EM "oracle". This formulation is convenient for our presentation because our improvements will mainly be focused on the EM procedure.

The guarantees of MWEM are established by Hardt et al. [2012]. For the runtime, each iteration takes $O(m|\mathcal{X}|)$ time, and there are $T = O(\alpha^{-2} \log m)$ iterations.

Theorem 3.1. *Suppose we are given a dataset $X \in \mathcal{X}^n$ with $n = \Omega(\varepsilon^{-1}\alpha^{-2} \log \frac{1}{\delta} \log \frac{m}{\beta} \sqrt{\log |\mathcal{X}|})$. The MWEM Algorithm is (ε, δ)–DP and outputs a synthetic dataset $\widehat{p} \in \Delta(\mathcal{X})$ such that with probability at least $1 - \beta$ it holds that*

$$||Q \cdot (\widehat{p} - h)||_\infty \leq \alpha.$$

Algorithm 1 MWEM Algorithm for Linear Queries

Inputs: Queries $Q = \{q_i\} \in [0,1]^{m \times |\mathcal{X}|}$, Dataset $X \in \mathcal{X}^n$ where $|X| = n$ with histogram h. Parameters $\alpha, \varepsilon, \delta > 0$.
Initialize $w^{(1)} \leftarrow \vec{1}^{|\mathcal{X}|}$
$T = 4\alpha^{-2} \ln m, \varepsilon_0 = \varepsilon(2T \ln \frac{1}{\delta})^{-1/2}$ and $\eta = \sqrt{\frac{\ln |\mathcal{X}|}{T}}$
Let $u(h, \hat{p}, q_i) := |\langle q_i, h\rangle - \langle q_i, \widehat{p}\rangle|$.
procedure MWU
  for $t = 1$ to T **do**
    $\widehat{p}^{(t)} \leftarrow w^{(t)} / \|w^{(t)}\|_1$
    $i_t \leftarrow \mathrm{EM}(\varepsilon_0/2,\, \widehat{p}^{(t)},\, h,\, u,\, \frac{1}{n})$
    $m_t \leftarrow \langle q_{i_t}, h\rangle + \mathrm{Lap}\left(\frac{2}{n\varepsilon_0}\right)$ ▷ private measurement
    $c^{(t)} \leftarrow \mathrm{sign}(m_t - \langle q_{i_t}, \widehat{p}^{(t)}\rangle) \cdot q_{i_t}$
    For each $x \in \mathcal{X}$: $w_x^{(t+1)} = w_x^{(t)} \cdot e^{\eta \cdot c_x^{(t)}}$
  Output $\widehat{p} := \frac{1}{T}\sum_{t=1}^{T} \widehat{p}^{(t)}$
procedure $\mathrm{EM}(\varepsilon_0, \hat{p}, h, u, \Delta_u)$
  return index $i \in [m]$ w.p. $\propto \exp\left(\frac{\varepsilon_0 \cdot u(h,\hat{p},q_i)}{2\Delta_u}\right)$.

The runtime of each iteration is $O(m|\mathcal{X}|)$.

3.3. **"Lazy" Exponential Mechanism.** To arrive at our improvement, we first recall that the exponential mechanism can be implemented using the classic Gumbel-Max-Trick (see Appendix D).

Lemma 3.2. *Let $x_1 \geq x_2 \geq \cdots \geq x_n$ be a list of real numbers, and define the distribution $p \in \Delta([n])$[1] where $p_i \propto \exp(x_i)$. Consider sampling n random variables $G_1, \ldots, G_n \sim \mathrm{Gumbel}(0,1)$ and let*

$$\hat{i} \in \arg\max_{i \in [n]} (x_i + G_i).$$

Then, $\hat{i}$ is distributed according to p.

Though this trick helps with numerical stability, sampling every G_i and taking the maximum still takes linear time. Mussmann et al. [2017] demonstrated that, if we possess knowledge of the $k = \sqrt{n}$ largest numbers $S_k = \{x_1, ..., x_k\}$, then sampling only $\Theta(\sqrt{n})$ Gumbel random variables suffices to sample from p. We will refer to this method as *lazy Gumbel Sampling* and we include more background information on it in Appendix E.

Computing S_k in general, however, also requires linear time. To get sublinear runtime, we require that the numbers have some specific structure. For example, when the x_i-s are inner products between a continuously evolving query vector q and a static vector dataset $V = \{v_1, ..., v_n\}$, we can obtain S_k by solving the *k-Maximum Inner Product Search* problem (k-MIPS) with dataset V and queries q. The k-MIPS problem is well-studied, particularly as it can be reduced to the k-NN problem [Neyshabur and Srebro, 2015]. For more information, see Appendix F. We note that *exact* k-MIPS has no known sublinear worst-case algorithm; however, *approximate* solvers such as LSH [Datar et al., 2004], IVF [Baranchuk et al., 2018], and HNSW [Malkov and Yashunin, 2018] achieve sublinear expected query time on typical inputs. Our framework already accounts for this: Theorems G.2 and G.10 in Section 3.5 give formal guarantees for Fast-MWEM when the MIPS oracle is only approximate.

[1]We use the $\Delta(\cdot)$ notation to denote the probability simplex over a domain.

Conveniently, this inner product structure arises very naturally in the MWEM settings we consider. For linear queries specifically, recall that our scores are indeed of the form $\langle q_i, h - \widehat{p} \rangle$, where $h - \widehat{p}$ is provided continuously by MWU and $Q = \{q_i\}_{i=1}^m$ are fixed in the beginning. Thus, we can first pre-process Q to build a k-MIPS index $\mathcal{H}$ and then query $\mathcal{H}$ in each iteration to obtain the top $\sqrt{m}$ inner products.

In this section we will state our algorithm in a manner agnostic to the implementation of index $\mathcal{H}$, though this will be a vital detail in our experimental section. For the theoretical analysis it will be enough to assume that each query to $\mathcal{H}$ to approximately retrieve S_k can be answered in $O(k)$ time. Indeed, the retrieval process itself can be lossy, a situation common to many fast k-MIPS algorithms, and we will specifically address the consequences of only *approximately* obtaining S_k in Section 3.5.

3.4. **Fast MWEM with a perfect index.** Following our previous discussion, we propose Fast-MWEM as Algorithm 2 for solving the linear query release problem. In that problem specifically, the scores in the exponential mechanism are actually the *absolute values* of inner products. To use a k-MIPS data structure we have to augment our query dataset by making it closed to complements: if $q_i \in Q$ then $1 - q_i \in Q$.

To aid presentation, we first analyze the algorithm under the assumption that the k-MIPS index $\mathcal{H}$ is *perfect*, meaning that it *exactly* finds the set S_k. We denote by $\mathcal{H}(k, h-p)$ the result of querying $\mathcal{H}$ with vector $h - p$ to find the query set $S_k \subset Q$ with the k-largest inner products $\langle q_i, h - p \rangle$.

Algorithm 2 Fast MWEM

Inputs: Queries $Q = \{q_i\}_{i=1}^m$, Dataset $X \in \mathcal{X}^n$ with histogram h. Parameters $\alpha, \varepsilon, \delta > 0$.
Initialize a k-MIPS index $\mathcal{H}$ on Q.
$T \leftarrow \frac{4 \ln m}{\alpha^2}, \varepsilon_0 \leftarrow \frac{\varepsilon}{\sqrt{T \cdot \ln \frac{1}{\delta}}}$, and $\eta = \sqrt{\frac{\ln |\mathcal{X}|}{T}}$.
procedure LazyEM($\varepsilon_0, p, h, \mathcal{H}, u, \Delta_u$)
    $S \leftarrow \mathcal{H}(\sqrt{m}, h - p)$.
    Sample $G_s \sim \text{Gumbel}(0, 1)$ for each $s \in S$.
    Let $M = \max_{s \in S} \left\{ \frac{\varepsilon_0 \cdot |\langle q_s, p - h \rangle|}{2\Delta_u} + G_s \right\}$
    Let $L = \min_{s \in S} \left\{ \frac{\varepsilon_0 \cdot |\langle q_s, p - h \rangle|}{2\Delta_u} \right\}$ and $B = M - L$
    Let $C \sim \text{Bin}(m - \sqrt{m}, 1 - e^{-e^{-B}})$
    Sample C queries from $\mathcal{Q} \setminus S$ into set T
    For $t \in T$, sample $U_t \sim \text{Uniform}(e^{-e^{-B}}, 1)$.
    Let $G_t = -\ln(-\ln(U_t))$.
    return $\arg\max_{i \in S \cup T} \left\{ \frac{\varepsilon_0 \cdot |\langle q_i, p - h \rangle|}{2\Delta_u} + G_i \right\}$

Referring back to Algorithm 1, we create $\mathcal{H}$ in the initialization stage, and substitute the *EM* algorithm with our lazy approach. The iterative MWU part remains identical, except that *LazyEM* is now invoked instead of EM.

We analyze Fast MWEM via the following theorem:

Theorem 3.3. *Let $\mathcal{H}$ be a randomized data structure built on a fixed dataset Q of size m, such that with probability at least $1 - \frac{1}{m}$ over its internal randomness, it exactly solves the k-MIPS problem correctly for* every *possible query vector q. Suppose that each query $\mathcal{H}(k, q)$ runs in $O(k)$ time.*

Let $n = \Omega(\log \frac{1}{\delta} \log \frac{m}{\beta} \sqrt{\log |\mathcal{X}|} \cdot \varepsilon^{-1} \alpha^{-2})$. Then Algorithm 2 is a $(\varepsilon, \delta + \frac{1}{m})$–DP algorithm that produces a synthetic dataset histogram $\widehat{p} \in \Delta(\mathcal{X})$ such that with probability at least $1 - \beta - \frac{1}{m}$, we have $||Q \cdot (\widehat{p} - h)||_\infty \leq \alpha$.

The expected per-iteration runtime is $\Theta(|\mathcal{X}|\sqrt{m})$.

Proof sketch; full proof in Appendix I. Let $\mathcal{G}$ be the event that $\mathcal{H}$ returns the exact top-$\sqrt{m}$ inner products for every query; $\Pr[\mathcal{G}] \geq 1 - \frac{1}{m}$ by assumption, and $\mathcal{G}$ is independent of the database.

Privacy: Conditioning on $\mathcal{G}$, each LazyEM call is distributionally identical to EM (Lemma 3.2 and Theorem E.1), so T sequential calls compose to (ε, δ)-DP via advanced composition (Theorem C.1); the $\frac{1}{m}$ failure probability of $\mathcal{H}$ contributes the extra slack.

Utility: On $\mathcal{G}$, LazyEM matches EM, so utility follows directly from Theorem 3.1.

Runtime: By Theorem E.1, each LazyEM call uses $O(\sqrt{m})$ Gumbel samples in expectation; combined with the $O(|\mathcal{X}|)$ MWU update, this gives $\Theta(|\mathcal{X}|\sqrt{m})$ per iteration. □

3.5. **Imperfect indices.** We now drop our simplifying assumption that the index $\mathcal{H}$ is perfect. A k-MIPS algorithm satisfying our $O(k)$ query runtime requirement can only provide an approximation to the correct set S_k, which we define as follows:

Definition 3.4. A set of indices S is an approximate top k if $|S| = k$ and for some constant c it holds that:

$$\max_{i \notin S} x_i - \min_{i \in S} x_i \leq c.$$

Let us assume that $\mathcal{H}(\sqrt{m}, p)$ only returns a c-approximate set S in Algorithm 2. Our analysis extends in two orthogonal directions depending on which guarantee one wishes to preserve (proofs in Appendix G).

Theorem 3.5 (**Preserving runtime**). *Using an approximate top-k set with error c, LazyEM runs in expected time $\Theta(\sqrt{m})$ per iteration and is $(\varepsilon + 2c)$-DP.*

Theorem 3.6 (**Preserving privacy**). *Using an approximate top-k set with error c, a variant of LazyEM that adjusts the margin B by c runs in expected time $e^c \cdot \Theta(\sqrt{m})$ per iteration and is ε-DP.*

Both proofs work by bounding how much the per-candidate sampling probabilities deviate from the exact mechanism: Theorem G.2 shows the ratio is within $e^{\pm c}$ for every candidate, directly yielding the privacy slack; Theorem G.10 recovers the exact distribution by shrinking the margin B by c, ensuring any winning candidate outside S is still sampled, at the cost of drawing more samples. In both cases, Fast-MWEM remains a sublinear-time, differentially private algorithm even when the index is imperfect.

4. Solving LPs Privately, Faster

In this section we apply our fast MWEM framework towards approximately solving Linear Programs (LPs) privately. Following [Hsu et al., 2014], we focus on solving LPs that have the form:

$$\begin{aligned} &\max_{x \in \mathbb{R}^d} c^\top x \\ \text{s.t.} \quad &Ax \leq b \end{aligned}$$

where $c \in \mathbb{R}^d$, $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$. Let $\mathcal{K}_{\text{OPT}} := \{x \in \mathbb{R}^d \mid c^\top x = \text{OPT}\}$. We can solve the original LP by repeatedly solving the following *feasibility* problem:

$$\begin{aligned} &\text{find } x \in \mathcal{K}_{\text{OPT}} \\ \text{s.t.} \quad &Ax \leq b \end{aligned}$$

and binary searching the value of OPT. Therefore, we focus only on feasibility LPs.

4.1. **Scalar Private LPs.** Consider a private database D that defines our problem parameters $A(D), b(D), c(D)$. Let

$$\rho = \sup_D ||A(D)||_\infty$$

be a publicly known upper bound to the *width* of our LP. In this section, we consider *low-sensitivity, scalar private* LPs where for neighboring databases D and D' it is true that $||b(D) - b(D')||_\infty \leq \Delta_\infty$. At the same time, $A(D) = A(D')$ and $c(D) = c(D')$. Other types of neighboring and sensitivity conditions can also be considered [Hsu et al., 2014]. Our method can be applied to those as well.

We further assume for simplicity that that the feasible solution is a distribution:

$$\begin{aligned} &\text{find } x \in \mathbb{R}_+^d \\ \text{s.t.} \quad &||x||_1 = 1 \text{ and } Ax \leq b \end{aligned}$$

Solving LPs approximately with the MWU method is a classic result: We start with the uniform distribution which we refine using the MWU rule. Every time we have a candidate solution $\widetilde{x}^{(t)}$, we find the worst constraint $p^{(t)}$, which is the row A_j maximizing $A_j\widetilde{x}^{(t)} - b_j$. Our losses are then the entries of that constraint.

To achieve privacy, we use the exponential mechanism to select the worst constraint with quality score

$$Q_t(i, b(D)) = A_i\widetilde{x}^{(t)} - b_i.$$

By Theorem C.1, our entire algorithm is (ε, δ)-DP. Analyzing the error of EM the solution x^* we obtain has $Ax^* \leq b + \alpha \cdot \vec{1}$ with probability at least $1 - \beta$ [Hsu et al., 2014], where

$$\alpha := \alpha_{\text{err}} = \widetilde{O}\left(\frac{\rho^{1/2}\Delta_\infty^{1/2}}{\varepsilon^{1/2}} \log^{1/4} d \log^{1/2} \frac{1}{\beta}\right).$$

Each iteration in this algorithm takes $\Theta(dm)$ time.

The improvement. Our fast-MWEM framework can improve this runtime via the familiar observation that scores can be written as inner products. Specifically, consider the vector set $\{A_i \circ b_i\}_{i=1}^m \in (\mathbb{R}^{d+1})^m$, where $\circ$ denotes concatenation. Then we can write:

$$Q_t(i, b(D)) = \langle A_i \circ b_i, \widetilde{x}^{(t)} \circ -1 \rangle.$$

We can then run Lazy-EM with queries $x'_t = \widetilde{x}^{(t)} \circ -1$.

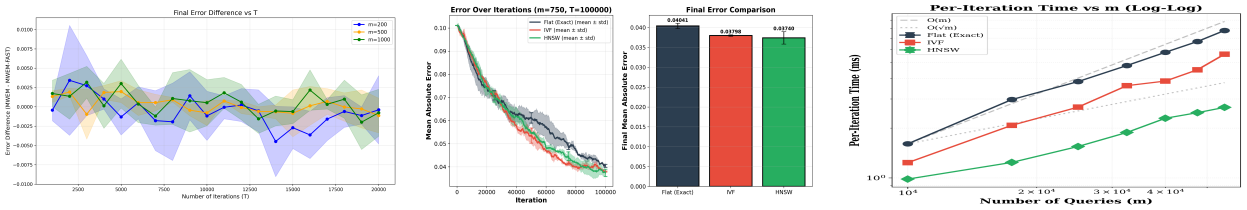
Theorem 4.1. *In the scalar-private, low-sensitivity setting, there exists an (ε, δ)-private algorithm for solving the feasibility LP $Ax \leq b$ that outputs a solution x^* such that with probability at least $1 - \beta$ we have $Ax^* \leq b + \alpha_{err} \cdot \vec{1}$. The runtime per iteration is $O(d\sqrt{m})$ in expectation.*

The full pseudocode is given as Algorithm 3 in Appendix A.

Constraint-private LPs. Fast-MWEM also applies to constraint-private LPs by combining the same lazy private-selection primitive with dense MWU on the dual. We defer the full construction and guarantee to Appendix B.1; at a high level, the dual oracle reduces to maximizing inner products against fixed column vectors, so preprocessing again yields an expected per-iteration runtime improvement from $O(md)$ to $O(m\sqrt{d})$.

5. Experiments

We validate that Fast-MWEM improves the runtime of MWEM while preserving accuracy and privacy, fixing $\varepsilon = 1$ and $\delta = 10^{-3}$ throughout. We use three k-MIPS implementations: a *flat* (exact linear-scan) baseline, *IVF*, and *HNSW*, sourced from the FAISS library [Douze et al., 2024] (see Appendix J for configuration). Additional ablation studies appear in Appendices K and L.



(A) Error diff: Fast-MWEM vs. MWEM.

(B) Error over iterations, all indices.

(C) Runtime vs. m, all indices.

FIGURE 1. Linear query release experiments. **(a)** Fast-MWEM error nearly matches MWEM; **(b)** all indices achieve similar accuracy; **(c)** HNSW and IVF achieve sublinear runtime scaling.

5.1. **Fast Private Linear Query Release.** We fix domain size $U = |\mathcal{X}| = 3000$, generate the true histogram from $n = 500$ samples of $\mathcal{N}(\frac{U}{3}, \frac{U}{15})$, and form each query as a binary vector with support sampled from $\mathcal{N}(\frac{U}{2}, \frac{U}{5})$.

Accuracy. For $m \in \{200, 500, 1000\}$ and $T = 20{,}000$ iterations, Figure 1a shows the error difference between Fast-MWEM (flat) and MWEM is consistently near zero, confirming our theoretical guarantee. Figure 1b shows all indices achieve similar error to the flat baseline, with error decreasing as T grows.

Runtime. Testing $10^4 \leq m \leq 10^5$, the flat index scales nearly linearly while IVF and HNSW give sublinear scaling (Figure 1c). HNSW yields the fastest runtime, matching the $O(\sqrt{m})$ prediction of our theory.

5.2. **Real-World Evaluation: ADULT Dataset.** To validate Fast-MWEM on a real-world distribution, we evaluate on the standard ADULT dataset ($n = 32{,}561$) for multi-way marginal query release. We set $T = 100$, $\varepsilon = 1.0$, and sample $m \in \{1{,}000, 10{,}000\}$ random 2- or 3-way marginal indicator queries. Table 1 shows that Fast-MWEM achieves a speedup of up to 42× on query selection while retaining competitive *mean* error. The elevated max error is expected: the approximate MIPS index occasionally misses the true worst-case query, which is precisely the approximation error c quantified in Theorems G.2 and G.10.

TABLE 1. Fast-MWEM vs. MWEM on the ADULT dataset ($T = 100$, $\varepsilon = 1.0$). "std" = standard MWEM, "fast" = Fast-MWEM (HNSW).

m	**Speedup (selection)**	**Mean error (std → fast)**	**Max error (std → fast)**
1,000	∼8.0×	∼0.021 → ∼0.035	∼0.166 → ∼0.398
10,000	∼41.9×	∼0.020 → ∼0.031	∼0.165 → ∼0.392

5.3. **Fast Scalar Private LP Solving.** We generate a feasibility LP with $A \sim \mathcal{N}(0, I_{m\times d})$, a solution $x^* \in \Delta([d])$, and solve $Ax \leq \beta$ where $\beta := Ax^* + \delta$ for a random perturbation δ. We fix $d = 20$, $\Delta_\infty = 0.1$, $\alpha = 0.5$.

Accuracy & Runtime. With $T = 5000$ and $3 \cdot 10^5 \leq m \leq 15 \cdot 10^5$, Fast-MWEM tracks MWEM's violated-constraint fraction nearly exactly, while HNSW vastly outperforms the flat index, scaling as $O(\sqrt{m})$; IVF does not yield significant speed-up in this regime. Full plots and additional experiments appear in Figure 4 and Appendix L.

6. Conclusion & Discussion

We introduced Fast-MWEM, a framework that reduces the per-iteration cost of MWEM's exponential mechanism from $O(m)$ to $\tilde{O}(\sqrt{m})$ in expectation. We applied this to private linear query release and private LP solving, preserving the privacy and utility guarantees of the originals.

Limitations. Our speedup also addresses only the m-dependence: the dependence on $|\mathcal{X}|$ is unchanged. Finally, IVF did not yield speed-ups in the LP setting, suggesting sensitivity to constraint geometry.

Future directions. Evaluating Fast-MWEM on real query workloads and combining it with domain-reduction heuristics (e.g., marginal-based synthetic data methods) could yield algorithms sublinear in both m and $|\mathcal{X}|$. The lazy-sampling primitive also extends naturally to other inner-product-based private selection problems, such as private boosting [Bun et al., 2020]. Tightening the approximation analysis to recover full (ε, δ)-DP without a runtime penalty would close the remaining gap between theory and practice.

Acknowledgements

This work originated as a final project for the course "Privacy in Statistics and Machine Learning" (Spring 2025) at Boston University, instructed by Professor Adam Smith. We thank Professor Smith for his guidance, encouragement, and insightful advice throughout the development of this research. We also wish to thank Ta Duy Nguyen for providing helpful comments on the manuscript and for directing our attention to the private linear programming (LP) solver problem.

References

Sergul Aydore, William Brown, Michael Kearns, Krishnaram Kenthapadi, Luca Melis, Aaron Roth, and Ankit A Siva. Differentially private query release through adaptive projection. In *International Conference on Machine Learning*, pages 457–467. PMLR, 2021.

Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 202–216, 2018.

Mark Bun, Marco Leandro Carmosino, and Jessica Sorrell. Efficient, noise-tolerant, and private learning via boosting. In *Conference on Learning Theory*, pages 1031–1077. PMLR, 2020.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st annual symposium on foundations of computer science*, pages 51–60. IEEE, 2010.

Cynthia Dwork, Nitin Kohli, and Deirdre Mulligan. Differential privacy in practice: Expose your epsilons! *Journal of Privacy and Confidentiality*, 9(2), 2019.

Khaled El Emam, Lucy Mosquera, and Richard Hoptroff. *Practical synthetic data generation: balancing privacy and the broad availability of data*. O'Reilly Media, 2020.

Alina Ene, Huy Le Nguyen, Ta Duy Nguyen, and Adrian Vladu. Solving linear programs with differential privacy. *arXiv preprint arXiv:2507.10946*, 2025.

Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. *Advances in neural information processing systems*, 25, 2012.

Mark Herbster and Manfred K Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1(Sep):281–309, 2001.

Justin Hsu, Aaron Roth, Tim Roughgarden, and Jonathan Ullman. Privately solving linear programs. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pages 612–624. Springer, 2014.

Haim Kaplan, Yishay Mansour, Shay Moran, Uri Stemmer, and Nitzan Tur. On differentially private linear algebra. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 2362–2373, 2025.

Terrance Liu, Giuseppe Vietri, and Steven Z Wu. Iterative methods for private synthetic data: Unifying framework and new methods. *Advances in Neural Information Processing Systems*, 34: 690–702, 2021.

Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

Ryan McKenna, Gerome Miklau, Michael Hay, and Ashwin Machanavajjhala. Hdmm: Optimizing error of high-dimensional statistical queries under differential privacy. *arXiv preprint arXiv:2106.12118*, 2021.

Ryan McKenna, Brett Mullins, Daniel Sheldon, and Gerome Miklau. Aim: An adaptive and iterative mechanism for differentially private synthetic data. *arXiv preprint arXiv:2201.12677*, 2022.

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103, 2007. doi: 10.1109/FOCS.2007.66.

Stephen Mussmann, Daniel Levy, and Stefano Ermon. Fast amortized inference and learning in log-linear models with randomly perturbed nearest neighbor search. *arXiv preprint arXiv:1707.03372*, 2017.

Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning*, pages 1926–1934. PMLR, 2015.

Aaron Roth and Tim Roughgarden. Interactive privacy via the median mechanism. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 765–774, 2010.

Jonathan Ullman and Salil Vadhan. Pcps and the hardness of generating private synthetic data. In *Theory of Cryptography Conference*, pages 400–416. Springer, 2011.

Appendix A. Fast Scalar Private LP Solver

Algorithm 3 Fast Scalar Private LP Solver

Input: $A \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$, error $\alpha \in (0,1)$, and privacy parameters $\varepsilon > 0, \delta \in (0,1), \Delta_\infty \in \mathbb{R}$.
procedure Initialization
  Let $\widetilde{x}^{(1)}$ be the uniform distribution in $\mathbb{R}^d$.
  Let $\rho := \max_{ij} |A_{ij}|$ be the width of the LP.
  Let $\alpha > 0$ be the desired accuracy.
  Set $T \leftarrow \frac{9\rho^2 \log d}{\alpha^2}$, $\eta \leftarrow \sqrt{\frac{\log d}{T}}$,
  $\varepsilon_0 \leftarrow \frac{\varepsilon}{\sqrt{8T\log(1/\delta)}}$, and $Q(i,b) = A_i\widetilde{x}^{(t)} - b_i$.
  Construct k-MIPS index $\mathcal{H}$ with $\{A_i \circ b_i\}_{i=1}^m$.
procedure MWU
  for $t = 1$ to T **do**
    Score: $Q_t(i, b(D)) := \langle A_i \circ b_i, \widetilde{x}^{(t)} \circ -1\rangle$
    $p^{(t)} = \text{LazyEM}(\varepsilon_0, -x'_t, 0, \mathcal{H}, Q, \Delta_\infty)$
    $\ell_i^{(t)} \leftarrow \frac{1}{\rho} A_{p^{(t)}i}$ for each $i \in [d]$.
    $X_i^{(t+1)} = e^{-\eta\ell_i^{(t)}} X_i^{(t)}$ for $i \in [d]$.
    $\widetilde{x}^{(t)} = \frac{X^{(t+1)}}{||X^{(t+1)}||_1}$.
  Output $\bar{x} = \frac{1}{T}\sum_{t=1}^T \widetilde{x}^{(t)}$

Appendix B. Dense Distributions and Bregman Projections

Definition B.1 (Dense Distributions)**.** A distribution y over $[n]$ is $\frac{1}{s}$-**dense** if $||y||_\infty \le \frac{1}{s}$.

Dense MWU requires projecting to the space of dense distributions in the following way:

Definition B.2 (Bregman Projections)**.** Let $F : [0,1]^n \to \mathbb{R}$ and $D_F(p,q) := F(p) - F(q) - \langle \nabla F(q), p - q\rangle$ be the F-Bregman divergence of p and q. We define the **Bregman projection** of $A \in [0,1]^n$ to the space of $\frac{1}{s}$-dense distributions as the solution to optimization problem

$$\Gamma_s A = \operatorname*{arg\,min}_{P \text{ is } \frac{1}{s}\text{-dense}} D_{\text{KL}}(P \,||\, A),$$

where $\text{KL}(x) = \sum_a x_a \log x_a$. Solving this via Lagrange Multipliers we obtain

$$\Gamma_s A_a = \frac{1}{s}\min\{1, cA_a\},$$

where c is such that $s = \sum_a \min\{1, cA_a\}$.

Lemma B.3 (Lemma 13 in [Hsu et al., 2014])**.** *Let A, A' be measures on $\mathcal{A}$ and $\mathcal{A} \cup a'$, identical on $\mathcal{A}$. Let $\widetilde{A}, \widetilde{A}'$ be the Bregman projections of A and A' into the set of $\frac{1}{s}$-dense distributions. Then,*

$$||\widetilde{A} - \widetilde{A}'||_1 \le \frac{1}{s}.$$

B.1. **Constraint-Private LPs via Dense MWU.** Another way to solve LPs approximately is to focus on the dual LP instead and apply MWU to compete with the best *dense* distribution in hindsight. This gives us a solver that finds an approximate solution x^*, though it might not satisfy a small fraction of the constraints. For background on dense distributions and Bregman projections, see Appendix B.

The Dense MWU algorithm operates on the dual space: instead of proposing a solution $\widetilde{x}^{(t)}$ and computing the worst constraint for $\widetilde{x}^{(t)}$, the algorithm suggests a distribution $\widetilde{y}^{(t)}$ over constraints and computes a solution x^* with minimal expected constraint violation. Computing x^* is done via the dual oracle:

Definition B.4. Let $\rho \geq \sup_D \sup_{x\in\mathbb{R}^d} ||A(D)x - b(D)||_\infty$ be a data-independent LP width. Given a distribution $y \in \Delta([m])$ over constraints, the (α, β)**- dual oracle** finds some candidate solution $x^* \in \mathbb{R}^d$ such that w.p. at least $1-\beta$,

$$\sum_{i=1}^m y_i(A_i x^*) \leq \min_{x\in\mathcal{K}} \sum_{i=1}^m y_i(A_i x) + \alpha \text{ and}$$
$$||Ax^* - b||_\infty \leq \rho.$$

After updating $\widetilde{y}^{(t)}$ via MWU, we project it to the space of $\frac{1}{s}$-dense distributions via the Bregman projection Γ_s at each step. It is known [Herbster and Warmuth, 2001] that this gives low regret guarantees against the best, in-hindsight, dense distribution:

Lemma B.5. *If $||\ell^{(t)}||_\infty \leq 1$ and $\eta \leq \frac{1}{2}$ then it is true that:*

$$\frac{1}{T}\sum_{t=1}^T \langle \ell^t, \widetilde{B}^t\rangle - \frac{1}{T}\sum_{t=1}^T \langle \ell^t, \widetilde{B}^*\rangle + \eta + \frac{\log n}{\eta T}$$

when $\widetilde{B}^*$ is any distribution uniform on a subset $S \subseteq [n]$ with $|S| \leq s$

In other words, we can get the same guarantee as the classic MWU, but while only using dense distributions!

[Hsu et al., 2014] prove that dense MWU finds $x \in \mathcal{K}_{\text{OPT}}$ such that *all but* $s-1$ constraints are violated with error at most α.

Constraint Privatization. In a constraint private LP, we assume that for neighboring datasets $D \sim D'$ the constraints $(A(D), b(D))$ and $(A(D'), b(D'))$ differ in that one of them has one extra row. Note that we no longer have a low-sensitivity assumption. Suppose our oracle is $\varepsilon' = \varepsilon/(2T\log(1/\delta))^{1/2}$ private when on neighboring instances y, y' we have

$$||y||_\infty, ||y'||_\infty \leq \frac{1}{s}, \text{ and } ||y - y'||_1 \leq \frac{2}{s}.$$

Then, by advanced composition the whole algorithm is (ε, δ)-private. When an extra row is added, it is known that the Bregman projections between y and y' are close (Lemma B.3), so privacy follows.

Fast-MWEM on the Dual Oracle. Our task becomes efficiently and privately implementing the dual oracle. Let us assume that the entries of matrix A, as well as the constraint c are positive (packing / covering LP). Recall that the oracle $\mathcal{O}$ wants to output, given some $y \in \Delta([m])$,

$$\arg\min_{x\in\mathcal{K}}\{y^T Ax\},$$

where $\mathcal{K} = \{x \in \mathbb{R}^d_+ : c^T x = \text{OPT}\}$. This is a linear program itself, so by the fundamental theorem of linear programming its solution has to be a vertex of the convex polytope $\mathcal{K}$. These vertices have

the form $v^{(j)} = \frac{\text{OPT}}{c_j} \cdot e_j$, so our optimization problem becomes

$$\text{find} \quad \arg\min_{j \in [d]} \left\{ \frac{\text{OPT}}{c_j} \cdot y^T A \vec{e_j} \right\}.$$

To solve it privately, we use the exponential mechanism with score function

$$Q(j, y) = -\frac{\text{OPT}}{c_j} y^T A_{:,j}.$$

If we let $N_j := -\frac{\text{OPT}}{c_j} \cdot (A^T)_j$ then we just have to maximize $\langle y, N_j \rangle$, over $j \in [d]$.

Note that the vectors N_j are fixed and can be preprocessed in the beginning of the algorithm. Every iteration we query our index $\mathcal{H}$ with y_j to implement LazyEM. For the proof, see Appendix H.

Theorem B.6. *If ρ is the width of our LP and $\alpha \leq 9\rho$, then there exists an algorithm that finds some x^* such that $A_i x^* \leq b_i + \alpha$ except for at most s constraints with*

$$s = \widetilde{O}\left(\frac{OPT^2 \log \frac{1}{\beta}}{c_{\min}^2 \alpha^2 \varepsilon} \right).$$

where c_{min} is the minimum value of c. The algorithm has an expected per-iteration runtime scaling with $O(m\sqrt{d})$, as opposed to the prior $O(md)$.

Appendix C. Properties of Differential Privacy

We use two key properties of DP in our work: adaptive composition and post-processing. Adaptive composition [Dwork et al., 2010] describes a situation when DP algorithms are linked sequentially in an adaptive way.

Theorem C.1. *Suppose algorithms $\mathcal{A}_1, ..., \mathcal{A}_k$ are (ε, δ)–DP. Let $\mathcal{A}'$ be the adaptive composition of these algorithms: on input database x, algorithm $\mathcal{A}_i$ is provided with x, and, for $i \geq 2$, with the output y_{i-1} of $\mathcal{A}_{i-1}$. Then, for any $\delta' \in (0, 1)$, Algorithm $\mathcal{A}$ is $(\widetilde{\varepsilon}, \widetilde{\delta})$–DP with*

$$\widetilde{\varepsilon} = \varepsilon \cdot \sqrt{2k \ln(1/\delta')} + 2k\varepsilon^2 \quad \text{and} \quad \widetilde{\delta} = k\delta + \delta'.$$

Post-processing guarantees that the applying a data-independent function to the output of a DP algorithm cannot degrade its privacy guarantees:

Theorem C.2. *Let $\mathcal{A} : U^n \to Y^m$ and $\mathcal{B} : Y^m \to Z^r$ be randomized algorithms, where U, Y, Z are arbitrary sets. If $\mathcal{A}$ is (ε, δ)–DP, then so is the composed algorithm $\mathcal{B}(\mathcal{A}(\cdot))$.*

Appendix D. The Gumbel Distribution and the Gumbel-Max-Trick

Suppose we want to sample an index $i \in [n]$ according to a score dataset $\{x_1, x_2, x_3, ..., x_n\}$ with probability $p_i \propto \exp(x_i)$, where p_i is the probability that we sample x_i. The distribution with

$$p_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

is often called a *categorical distribution*, and the Gumbel Max Trick gives us a numerically stable way to sample from it. We begin by defining the Gumbel distribution and reviewing some of its properties.

Definition D.1 (The Gumbel Distribution)**.** Let $X \sim \text{Gumbel}(\mu, \beta)$. Then, X has expectation of $\mu + \gamma'\beta$, where $\gamma' \approx 0.577$ is the Euler's constant, has variance of $\frac{\pi^2}{6}\beta^2$, and follows the PDF and CDF

$$f_X(x) = \frac{1}{\beta}\exp(-z - \exp(-z)) \text{ and}$$
$$F_X(x) = \exp(-\exp(-z)),$$

respectively, for all $x \in \mathbb{R}$, where $z = \frac{x-\mu}{\beta}$.

Lemma D.2. *Let $x_1, x_2, x_3, ..., x_n$ be real numbers, and define the distribution $p \in \Delta([n])$, where*

$$p_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}.$$

Consider sampling n Gumbel random variables $G_1, \ldots, G_n \sim \text{Gumbel}(0, 1)$ and let

$$\hat{i} \in \arg\max_{i \in [n]}(x_i + G_i).$$

Then, $\hat{i}$ is distributed according to p.

Proof. Let q_i be the probability that $\hat{i} = i$ for all $i \in [n]$. By symmetry, we only need to prove that $q_1 \propto \exp(x_1)$.

Fix $G_1 = x$. Then, by the Gumbel distribution, $f_{G_1}(x) = e^{-x-e^{-x}}$. For $\hat{i}$ to be 1, we need $G_i < x_1 + G_1 - x_i$ for $\forall i \in [n] \setminus \{1\}$. Thus, the probability that $G_i < x_1 + G_1 - x_i$ is $e^{-e^{x_i - x_1 - G_1}}$.

Then, by law of total probability,

$$\begin{aligned}
\Pr[\hat{i} = 1] &= \int_{-\infty}^{\infty} f_{G_i}(x) \prod_{i=2}^{n} \Pr[G_i < x] dx \\
&= \int_{-\infty}^{\infty} e^{-x-e^{-x}} \prod_{i=2}^{n} e^{-e^{x_i - x_1 - x}} dx \\
&= \int_{-\infty}^{\infty} e^{x_1 - u - e^{x_1 - u}} \prod_{i=2}^{n} e^{-e^{x_i - u}} du && \text{(Substitute } u = x + x_1\text{)} \\
&= e^{x_1} \int_{-\infty}^{\infty} e^{-u} \prod_{i=1}^{n} e^{-e^{x_i - u}} du \\
&= e^{x_1} C. && \text{(For some constant } C\text{)}
\end{aligned}$$

Therefore, $q_1 \propto \exp(x_1)$, and $\hat{i}$ is distributed according to p. □

Note also the following property of the Gumbel distribution:

Lemma D.3. *The probability distribution of G_1 where $G_1 \sim \text{Gumbel}(0, 1)$ conditioned on $G_1 > B$ is the same as the probability distribution of G_2 where $G_2 = -\ln(-\ln(U))$ and $U \sim \text{Uniform}(\exp(-\exp(-B)), 1)$.*

Proof. We will prove that the distributions of G_1 and G_2 are the same by proving that their CDFs are the same namely, $F_{G_1}(x) = F_{G_2}(x)$ for all $x \in \mathbb{R}$.

First, we will determine $F_{G_1}(x)$. When $x \leq B$, $F_{G_1}(x) = 0$. When $x > B$, $F_{G_1}(x) = \frac{\Pr[G_1 \leq x]}{\Pr[G_1 > B]} = \frac{\Pr[G_1 \leq x]}{1 - \Pr[G_1 \leq B]} = \frac{e^{-e^{-x}}}{1 - e^{-e^{-B}}}$. Therefore,

$$F_{G_1}(x) = \begin{cases} 0; & x \leq B \\ \frac{e^{-e^{-x}}}{1 - e^{-e^{-B}}}; & B < x \end{cases}$$

Next, we will determine $F_{G_2}(x)$. Since $U \sim \text{Uniform}(\exp(-\exp(-B)), 1)$, then

$$F_U(x) = \begin{cases} 0; & x \leq e^{-e^{-B}} \\ \frac{x - e^{-e^{-B}}}{1 - e^{-e^{-B}}}; & e^{-e^{-B}} < x \leq 1 \\ 1; & 1 < x \end{cases}$$

After that, we substitute U with G_2 and x with $e^{-e^{-x}}$ yielding

$$F_{G_2}(x) = \begin{cases} 0; & e^{-e^{-x}} \leq e^{-e^{-B}} \\ \frac{e^{-e^{-x}} - e^{-e^{-B}}}{1 - e^{-e^{-B}}}; & e^{-e^{-B}} < e^{-e^{-x}} \leq 1 \\ 1; & 1 < e^{-e^{-x}}. \end{cases}$$

Simplifying all expressions result in $F_{G_1}(x) = F_{G_2}(x)$.

□

Appendix E. Lazy Gumbel Sampling

In the Gumbel Max Trick of Lemma D.2 above, we have to calculate n values of $x_i + G_i$ but since it is very rare that a Gumbel noise will be high, it is very unlikely that we will choose an index i such that x_i is small. Hence, at least intuitively, we often do not need to sample all n Gumbel noise random variables. The following algorithm, due to [Mussmann et al., 2017], shows how to perform Gumbel Max Trick while only using $\Theta(\sqrt{n})$ samples in expectation.

Algorithm 4 Lazy Gumbel Sampling, [Mussmann et al., 2017]

Inputs: An integer $k \in [n]$, and a score dataset $X = \{x_1, x_2, x_3, ..., x_n\} \in \mathbb{R}^n$
Let S be the set of the indices of k largest x_i from X
for $j \in S$ **do**
    Sample $G_j \sim \text{Gumbel}(0, 1)$
Let $M = \max_{j \in S}(x_j + G_j)$, $m = \min_{j \in S} x_j$, and $B = M - m$
Let $C \sim \text{Bin}(n - k, 1 - \exp(-\exp(-B)))$
Let $T \subseteq [n] \setminus S$ be the set of C distinct indices sampled uniformly from $[n] \setminus S$.
Sample $U_i \sim \text{Uniform}(\exp(-\exp(-B)), 1)$, and let $G_i = -\ln(-\ln(U_i))$ for $\forall i \in T$.
▷ This simulates $G_i \sim \text{Gumbel}(0, 1)$ conditioned on $G_i > B$
return $\arg\max_{i \in S \cup T}(x_i + G_i)$

This is formalized in the following theorem of [Mussmann et al., 2017]:

Theorem E.1 ([Mussmann et al., 2017])**.** *The Lazy Gumbel Algorithm on a score dataset* $\{x_1, ..., x_n\}$ *outputs an index* $i \in [n]$ *with probability:*

$$p_i \propto \exp(x_i)$$

using $\Theta(\sqrt{n})$ *samples in expectation when* $k \approx \Theta(\sqrt{n})$.

Appendix F. From k-MIPS to k-Nearest Neighbor Search

The regime we are considering is when the scores x_i are given by inner products of a query vector $q \in \mathbb{R}^d$ with a set of key vectors $\{k_1, ..., k_n\} \subseteq \mathbb{R}^d$. In other words, $x_i = \langle q, k_i \rangle$. In addition, we are considering a situation in which we have to sample from m categorical distributions, defined by query vectors $\{q_1, ..., q_m\}$ on a fixed set of key vectors. In this setting, identifying the top k inner

products can be done efficiently, in total time $O(n + m\sqrt{n})$ rather than $O(nm)$, which is where our improvement lies.

More specifically, we recognize that this is exactly the k–MIPS (Maximum Inner Product Search) problem. This problem has been studied extensively, and we can use any data structure developed for it to solve it. By initializing a data structure like this on the set $\{k_1, ..., k_n\}$ and then querying it for each incoming query q_i, we get the practical improvement we seek.

We can also make use of the observation that this problem is reducible to the k–nearest neighbor problem, which is also very well studied and for which there are also a wide variety of developed algorithms. We elaborate on this briefly below:

We know that

$$q^T k_i = \frac{1}{2}(||q||_2^2 + ||k_i||_2^2 - ||q - k_i||_2^2)$$

and $||q||_2^2$ is a constant for all $i \in [n]$. Therefore, if $||k_i||_2^2$ is also a constant for all $i \in [n]$, then $q^T k_i$ will be largest when $||q - k_i||_2^2$ is smallest. Thus, we will be able to reduce MIPS to kNN.

Let M be an arbitrary number that is known to be at least $\max_{i\in[n]} ||k_i||_2^2$. Then, we can transform k_i to

$$\left[k_i^T, \sqrt{M - ||k_i||_2^2}\right]^T$$

so that $||k_i||_2^2 = M$ for all $i \in [n]$. Next, we transform q to $[q^T, 0]^T$ so that it has the same dimension as k_i. Since this transformation preserves the dot product $\langle q, k_i\rangle$, we have successfully reduced MIPS to kNN.

Appendix G. Robustness to Approximation

In this section, we show that if our k-MIPS algorithm only retrieves the *approximate top-k* inner products, we can still perform our improved algorithm.

Definition G.1. A set of indices S is an approximate top k if $|S| = k$ and

$$\max_{i\notin S} x_i - \min_{i\in S} x_i \leq c,$$

for some constant c.

G.1. Preserving Runtime.

Here, we describe how to preserve the $\Theta(\sqrt{n})$ runtime under approximation, with some diminished privacy guarantees.

Algorithm 5 Lazy Gumbel Sampling using Approximate Top-k while preserving runtime

Inputs: An integer $k \in [n]$, and a score dataset $X = \{x_1, x_2, x_3, ..., x_n\} \in \mathbb{R}^n$
Let S be an **approximate top-k** of X.
for $j \in S$ **do**
    Sample $G_j \sim \text{Gumbel}(0, 1)$
Let $M = \max_{j\in S}(x_j + G_j)$, $m = \min_{j\in S} x_j$, and $B = M - m$
Let $C \sim \text{Bin}(n - k, 1 - \exp(-\exp(-B)))$
Let $T \subseteq [n] \setminus S$ be the set of C distinct indices sampled from $[n] \setminus S$.
Sample $U_i \sim \text{Uniform}(\exp(-\exp(-B)), 1)$, and let $G_i = -\ln(-\ln(U_i))$ for every $i \in T$.
return $\arg\max_{i\in S\cup T}(x_i + G_i)$

Theorem G.2. *Given the set S of the approximate top-k scores of dataset X in the exponential mechanism with candidate set $[n]$, Algorithm 5 runs in expected time $\Theta(\sqrt{n})$ and is $(\varepsilon + 2c)$-DP.*

We split the proof of Theorem G.2 into two lemmas: Lemma G.3 proves the runtime guarantees and Lemma G.4 proves the privacy guarantees.

We first prove the runtime guarantees.

Lemma G.3. *Algorithm 5 runs in expected time* $\Theta(\sqrt{n})$.

Proof. The proof is the same as the proof of Theorem E.1 since the upper bound on the expected number of samples is the same regardless of the choice of S. □

Next, to prove that Algorithm 5 is $(\varepsilon + 2c)$-DP, we first prove that the probability of the index $i \in [n]$ returned by Algorithm 5 is bounded above and below a constant factor of the true top-k probability. Let $\mathcal{I}$ and $\mathcal{I}'$ be Algorithm 4 and Algorithm 5, respectively, and $\mathcal{I}(X)$ and $\mathcal{I}'(X, S)$ be the outputs of Algorithm 4 on dataset X and Algorithm 5 on dataset X with approximate top-k set S, respectively.

Lemma G.4. *For any score dataset* $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^n$, *and every* $i \in [n]$, *we have*

$$e^{-c} \Pr[\mathcal{I}(X) = i] \leq \Pr[\mathcal{I}'(X, S) = i] \leq e^{c} \Pr[\mathcal{I}(X) = i].$$

Let S^* be the set of true top-k indices. Define sets $P = S$, $Q = S^* \setminus S$, and $R = [n] \setminus (S \cup S^*)$. Furthermore, define

$$A_P = \sum_{i \in P} e^{x_i}, \qquad A_Q = \sum_{i \in Q} e^{x_i}, \qquad A_R = \sum_{i \in R} e^{x_i}.$$

We split the proof of Lemma G.4 into 5 claims: Claim G.5 and Claim G.8 prove the lower bound of $\Pr[\mathcal{I}'(X, S) = i]$, and Claim G.6, Claim G.7, and Claim G.9 prove the upper bound of $\Pr[\mathcal{I}'(X, S) = i]$.

First, we prove a lower bound on $\Pr[\mathcal{I}'(X, S) = i]$, if $i \notin Q$.

Claim G.5. *For an index* $i \in [n]$, *if* $i \notin Q$, *then* $\Pr[\mathcal{I}(X) = i] \leq \Pr[\mathcal{I}'(X, S) = i]$.

Proof. Consider the event $\mathcal{I}'(X, S) = i$. If $i = \arg\max_{j \in [n]}\{x_j + G_j\}$, then i will be returned by $\mathcal{I}$ because of the Gumbel Max Trick and by $\mathcal{I}'$ because either $i \in S$ or $G_i > B$. If $i \neq \arg\max_{j \in [n]}\{x_j + G_j\}$, there exists some index $j \in Q$ with greater score value than i, but with noise value less than B, which means that i can still be sampled by $\mathcal{I}'$, though not by $\mathcal{I}$. Thus, if $i \notin Q$, we have $\Pr[\mathcal{I}'(X, S) = i] \geq \Pr[\mathcal{I}(X) = i]$. □

The next two claims prove an upper bound on $\Pr[\mathcal{I}'(X, S) = i]$, if $i \in Q$.

Claim G.6. *For an index* $i \in [n]$, *if* $Q = \{i\}$, *then we have* $\Pr[\mathcal{I}'(X, S) = i] \leq \Pr[\mathcal{I}(X) = i]$.

Proof. Suppose for contradiction that $\Pr[\mathcal{I}'(X, S) = i] > \Pr[\mathcal{I}(X) = i]$.

By Claim G.5, for all $j \in [n] \setminus \{i\}$, we have $\Pr[\mathcal{I}'(X, S) = j] \geq \Pr[\mathcal{I}(X) = j]$. Then we have

$$\begin{aligned}
\sum_{j \in [n]} \Pr[\mathcal{I}'(X, S) = j] &= \Pr[\mathcal{I}'(X, S) = i] + \sum_{j \in [n] \setminus \{i\}} \Pr[\mathcal{I}'(X, S) = j] \\
&\geq \Pr[\mathcal{I}'(X, S) = i] + \sum_{j \in [n] \setminus \{i\}} \Pr[\mathcal{I}(X) = j] \\
&> \sum_{j \in [n]} \Pr[\mathcal{I}(X) = j] = 1,
\end{aligned}$$

a contradiction. □

Claim G.7. *For an index $i \in [n]$, if $i \in Q$ and $|Q| > 1$, then we have* $\Pr[\mathcal{I}'(X,S) = i] \leq e^c \Pr[\mathcal{I}(X) = i]$.

Proof. Consider a modified dataset $\widetilde{X} = \{\widetilde{x}_1, ..., \widetilde{x}_n\} \in \mathbb{R}^n$, where for all $j \in [n]$, we have

$$\widetilde{x}_j = \begin{cases} x_j - c & \text{if } j \in Q \setminus \{i\} \\ x_j & \text{otherwise} \end{cases},$$

and let $\widetilde{Q}$ be the analogous set to Q, but for dataset $\widetilde{X}$.

We first prove that $\Pr[\mathcal{I}'(X,S) = i] \leq \Pr[\mathcal{I}'(\widetilde{X},S) = i]$. To prove this statement, we fix the Gumbel noise G_i on index i. Then, we have

$$\Pr[\mathcal{I}'(X,S) = i \mid G_i] = \prod_{j \in Q \setminus \{i\}} \Pr[\underbrace{G_j < \max\{x_i + G_i - x_j, B\}}_{(1)} \mid G_i] \cdot \prod_{j \notin Q} \Pr[G_j < x_i + G_i - x_j \mid G_i]$$

(for every $j \in Q \setminus \{i\}$, we have $\mathcal{I}'(X,S) \neq j$ if $x_j + G_j < x_i + G_i$ or $G_j < B$)

$$\leq \prod_{j \in Q \setminus \{i\}} \Pr[\underbrace{G_j < \max\{x_i + G_i - (x_j - c), B\}}_{(2)} \mid G_i] \cdot \prod_{j \notin Q} \Pr[G_j < x_i + G_i - x_j \mid G_i]$$

(event (1) is a subset of event (2))

$$\leq \prod_{j \in Q \setminus \{i\}} \Pr[G_j < \max\{\widetilde{x}_i + G_i - \widetilde{x}_j, B\} \mid G_i] \cdot \prod_{j \notin Q} \Pr[G_j < \widetilde{x}_i + G_i - \widetilde{x}_j \mid G_i]$$
$$= \Pr[\mathcal{I}'(\widetilde{X},S) = i \mid G_i].$$

Then, by the law of total probability, we have

$$\begin{aligned} \Pr[\mathcal{I}'(X,S) = i] &= \int_{x \in \mathbb{R}} \Pr[\mathcal{I}'(X,S) = i \mid G_i = x] \cdot f_{G_i}(x) dx \\ &\leq \int_{x \in \mathbb{R}} \Pr[\mathcal{I}'(\widetilde{X},S) = i \mid G_i = x] \cdot f_{G_i}(x) dx \\ &= \Pr[\mathcal{I}'(\widetilde{X},S) = i]. \end{aligned}$$

Since we decrease all x_j where $j \in Q \setminus \{i\}$ by c, these elements will no longer be in the true top-k. Therefore, $|\widetilde{Q}| = 1$.

Since $|\widetilde{Q}| = 1$, we have

$$\begin{aligned} \text{(by Claim G.6)} \qquad \Pr[\mathcal{I}'(\widetilde{X},S) = i] &\leq \Pr[\mathcal{I}(\widetilde{X}) = i] \\ &= \frac{e^{x_i}}{A_P + e^{-c}(A_Q - e^{x_i}) + e^{x_i} + A_R} \\ &= \frac{e^c e^{x_i}}{e^c A_P + A_Q + (e^c - 1)e^{x_i} + e^c A_R} \\ &\leq \frac{e^c e^{x_i}}{A_P + A_Q + A_R} \\ &= e^c \Pr[\mathcal{I}(X) = i], \end{aligned}$$

which proves Claim G.7. □

Next, we prove a lower bound on $\Pr[\mathcal{I}'(X,S) = i]$, if $i \in Q$.

Claim G.8. *For an index $i \in [n]$, if $i \in Q$, then we have $e^{-c}\Pr[\mathcal{I}(X) = i] \leq \Pr[\mathcal{I}'(X, S) = i]$.*

Proof. Again, consider a modified dataset $\widetilde{X} = \{\widetilde{x}_1, ..., \widetilde{x}_n\}$, where for all $j \in [n]$, we have

$$\widetilde{x}_j = \begin{cases} x_j + c & \text{if } j \in S \\ x_j & \text{otherwise} \end{cases},$$

and let $\widetilde{S}^*$ be the analogous set to S^*, but for dataset $\widetilde{X}$. Then, we have $S = \widetilde{S}^*$.

First we prove that $\Pr[\mathcal{I}'(\widetilde{X}, S) = i] \leq \Pr[\mathcal{I}(X, S) = i]$. Consider fixing $G_1, ..., G_n$, such that $\mathcal{I}'(\widetilde{X}, S) = i$ is true. Then we know that $\widetilde{x}_i + G_i = \max_{j\in[n]}\{\widetilde{x}_j + G_j\}$. Then, this implies that $x_i + G_i = \max_{j\in[n]}\{x_j + G_j\}$, since

$$\begin{aligned}
\widetilde{x}_i + G_i &= \max_{j\in[n]}\{\widetilde{x}_j + G_j\} \\
&= \max\{\max_{j\in S}\{(x_j + c) + G_j\}, \max_{j\notin S}\{x_j + G_j\}\} \\
&\geq \max\{\max_{j\in S}\{x_j + G_j\}, \max_{j\notin S}\{x_j + G_j\}\} \\
&= \max_{j\in[n]}\{x_j + G_j\}.
\end{aligned}$$

Thus, the event $\widetilde{x}_i + G_i = \max_{j\in[n]}\{\widetilde{x}_j + G_j\}$ is a subset of $x_i + G_i = \max_{j\in[n]}\{x_j + G_j\}$, and

$$\Pr[\mathcal{I}'(\widetilde{X}, S) = i] \leq \Pr[\mathcal{I}(X, S) = i].$$

Next, we prove $e^{-c}\Pr[\mathcal{I}(X) = i] \leq \Pr[\mathcal{I}'(\widetilde{X}, S) = i]$.

Since $S = \widetilde{S}^*$, for all $i \in [n]$, we have $\Pr[\mathcal{I}'(\widetilde{X}, S) = i] = \Pr[\mathcal{I}(\widetilde{X}) = i]$. We then have

(Since we added c to every element in S)

$$\begin{aligned}
\Pr[\mathcal{I}(\widetilde{X}) = i] &= \frac{e^{x_i}}{e^c A_P + A_Q + A_R} \\
&\geq e^{-c}\frac{e^{x_i}}{A_P + A_Q + A_R} \\
&= e^{-c}\Pr[\mathcal{I}(X) = i].
\end{aligned}$$

Thus, we have

$$e^{-c}\Pr[\mathcal{I}(X) = i] \leq \Pr[\mathcal{I}'(\widetilde{X}, S) = i] \leq \Pr[\mathcal{I}(X, S) = i],$$

which proves Claim G.8 □

Finally, we prove an upper bound on $\Pr[\mathcal{I}(X, S) = i]$, if $i \notin Q$.

Claim G.9. *For index $i \in [n]$, if $i \notin Q$, then $\Pr[\mathcal{I}(X, S) = i] \leq e^c \Pr[\mathcal{I}(X) = i]$.*

Proof. Let $p = \sum_{j\in Q}\Pr[\mathcal{I}'(X, S) = j]$. Then, by Claim G.8, we have

$$p \geq e^{-c}\frac{A_Q}{A_P + A_Q + A_R}.$$

Consider the probability $\Pr[\mathcal{I}'(X, S) = i]$. We have

$$\begin{aligned}
\Pr[\mathcal{I}'(X, S) = i] &= \Pr[\mathcal{I}'(X, S) = i \mid \mathcal{I}'(X, S) \notin Q] \cdot \Pr[\mathcal{I}'(X, S) \notin Q] \\
&= \frac{e^{x_i}}{A_P + A_R} \cdot (1 - p).
\end{aligned}$$

Consider any element $a \in P$ and $b \in Q$. We have

$$x_b - x_a \le \max_{j \in Q} x_j - \min_{j \in P} x_j \le c \iff e^{x_b} \le e^{c + x_a},$$

which implies

$$A_Q = \sum_{j \in Q} e^{x_j} \le \sum_{j \in P} e^{c + x_j} = e^c A_P,$$

where the inequality holds because $|P| = k \ge |Q|$.

Since $p \ge \frac{e^{-c} A_Q}{A_P + A_Q + A_R}$, we have $1 - p \le \frac{A_P + (1 - e^{-c}) A_Q + A_R}{A_P + A_Q + A_R}$. Then, we have

$$\begin{aligned}
\Pr[\mathcal{I}'(X, S) = i] &= (1 - p) \cdot \frac{e^{x_i}}{A_P + A_R} \\
&\le \frac{A_P + (1 - e^{-c}) A_Q + A_R}{A_P + A_Q + A_R} \cdot \frac{e^{x_i}}{A_P + A_R} \\
&= \frac{A_P + (1 - e^{-c}) A_Q + A_R}{A_P + A_R} \cdot \frac{e^{x_i}}{A_P + A_Q + A_R} \\
&\le \frac{A_P + (1 - e^{-c}) e^c A_P + A_R}{A_P + A_R} \cdot \frac{e^{x_i}}{A_P + A_Q + A_R} \\
&= \frac{e^c A_P + A_R}{A_P + A_R} \cdot \frac{e^{x_i}}{A_P + A_Q + A_R} \\
&\le e^c \cdot \frac{e^{x_i}}{A_P + A_Q + A_R} \\
&= e^c \Pr[\mathcal{I}(X) = i],
\end{aligned}$$

which proves Claim G.9 □

We can now combine the above claims to prove Lemma G.4.

Proof of Lemma G.4. Combining Claim G.5, Claim G.6, Claim G.7, Claim G.8, and Claim G.9 altogether proves $e^{-c} \Pr[\mathcal{I}(X) = i] \le \Pr[\mathcal{I}'(X, S) = i] \le e^c \Pr[\mathcal{I}(X) = i]$. □

We are now ready to prove Theorem G.2.

Proof of Theorem G.2. We first prove that Algorithm 5 is $(\varepsilon + 2c)$-DP.

Let Ω_S be the sample space of all possible approximate top-k sets, and $\mathcal{I}'(X)$ be the output of the index, which includes the random coins of the selection of $S \sim \Omega_S$. Then, for all $T \subseteq [n]$, we have

$$\begin{aligned}
\Pr[\mathcal{I}'(X) \in T] &= \sum_{S \in \Omega_S} \sum_{i \in T} \Pr[S] \cdot \Pr[\mathcal{I}'(X, S) = i] \\
\text{(by Lemma G.4)} \qquad &\le \sum_{S \in \Omega_S} \sum_{i \in T} \Pr[S] \cdot e^c \Pr[\mathcal{I}(X) = i] \\
&= \sum_{S \in \Omega_S} \Pr[S] \cdot e^c \Pr[\mathcal{I}(X) \in T] \\
&= e^c \Pr[\mathcal{I}(X) \in T].
\end{aligned}$$

Next, let X' be the neighboring dataset of X. We will prove that $\Pr[\mathcal{I}'(X) \in T] \leq e^{\varepsilon+2c} \Pr[\mathcal{I}'(X') \in T]$. We have

$$\begin{aligned} \Pr[\mathcal{I}'(X) \in T] &\leq e^{c} \Pr[\mathcal{I}(X) \in T] \\ \text{(Algorithm 4 is } \varepsilon\text{-DP)} \qquad &\leq e^{\varepsilon+c} \Pr[\mathcal{I}(X') \in T] \\ &\leq e^{\varepsilon+2c} \Pr[\mathcal{I}'(X') \in T]. \end{aligned}$$

Similarly, we have $e^{-\varepsilon-2c} \Pr[\mathcal{I}'(X') \in T] \leq \Pr[\mathcal{I}'(X) \in T]$ for all $T \subseteq [n]$. Combining Lemma G.3, this concludes the proof that Algorithm 5 is $(\varepsilon + 2c)$-DP and runs in expected time $\Theta(\sqrt{n})$. □

G.2. **Preserving Privacy.** Here, we describe our algorithm for preserving privacy under approximation, at the cost of a slightly higher runtime.

The privacy preserving algorithm is identical to Algorithm 5, except we now set $B = M - m - c$.

Algorithm 6 Lazy Gumbel Sampling using Approximate Top-k while preserving runtime

Inputs: An integer $k \in [n]$, and a score dataset $X = \{x_1, x_2, x_3, ..., x_n\} \in \mathbb{R}^n$
Let S be an **approximate top-k** of X
for $j \in S$ **do**
    Sample $G_j \sim \text{Gumbel}(0, 1)$
Let $M = \max_{j \in S}(x_j + G_j)$, $m = \min_{j \in S} x_j$, and $B = M - m - c$
Let $C \sim \text{Bin}(n - k, 1 - \exp(-\exp(-B)))$
Let $T \subseteq [n] \setminus S$ be the set of C distinct indices sampled from $[n] \setminus S$.
Sample $U_i \sim \text{Uniform}(\exp(-\exp(-B)), 1)$, and let $G_i = -\ln(-\ln(U_i))$ for $\forall i \in T$.
return $\arg\max_{i \in S \cup T}(x_i + G_i)$

Theorem G.10. *Given the set S, the approximate top-k scores of dataset X, Algorithm 6 runs in expected time $e^c \cdot \Theta(\sqrt{n})$ and is ε-DP.*

Proof. We first prove that Algorithm 6 is ε-DP.

Given our approximate top-k set S, we have that $\max_{i \notin S} x_i - \min_{i \in S} x_i \leq c$. For an element $j \notin S$ to achieve $j = \arg\max_{i \in [n]}\{x_i + G_i\}$, the Gumbel noise G_j need to be at least

$$\begin{aligned} \max_{i \in S}\{x_i + G_i\} - x_j &\geq \max_{i \in S}\{x_i + G_i\} - \max_{i \notin S}\{x_i\} \\ &\geq \max_{i \in S}\{x_i + G_i\} - \min_{i \in S}\{x_i\} - c \\ &= M - m - c \\ &= B. \end{aligned}$$

Therefore, we guarantee that if $j \notin S$ and $G_j < B$, index j cannot be chosen. Thus, Algorithm 6 follows the same distribution as exponential mechanism and has the same privacy guarantees, which is ε-DP.

It is also proven by Mussmann et al. [Mussmann et al., 2017] that $\mathbb{E}[C] \leq \frac{e^c n}{k}$. Therefore, when $k = \Theta(\sqrt{n})$, we need to sample $\sqrt{n}$ maximum score from indices $[n]$ and at most $\frac{e^c n}{k} = e^c\sqrt{n}$ extra indices from $[n] \setminus S$ in expectation. Thus, we need to sample $e^c \cdot \Theta(\sqrt{n})$ elements in expectation.

Thus, Algorithm 6 runs in expected time $e^c \cdot \Theta(\sqrt{n})$ and is ε-DP. □

Appendix H. Proof of Theorem B.6

We rely on the following lemma by Hsu et al. [2014]:

Lemma H.1. *Let $0 \le \alpha \le 9\rho$ and $\beta \in (0,1)$. With probability at least $1-\beta$, dense MWU with density s and equipped with a $(\frac{\alpha}{3}, \frac{\beta}{T})$–approximate, ρ-bounded dual oracle finds some $x^* \in \mathcal{K}_{OPT}$ for which $A_i x^* \le b_i + \alpha$ except for at most $s-1$ constraints.*

Our algorithm effectively implements the exponential mechanism. Therefore, our dual oracle has width:

$$\rho = \frac{\mathrm{OPT}}{c_{\min}} - b_{max}$$

where $c_{\min} := \min_{i=1}^{d} c_i$ and $b_{\max} = \max_{i=1}^{d} b_i$. Recall the quality scores $Q(j,y) = -\frac{\mathrm{OPT}}{c_j} y^T A_{:,j}$. The sensitivity is at most $\frac{3\mathrm{OPT}}{c_{\min} s}$ because y can change up to $\frac{2}{s}$ (Lemma B.3) and a new constraint may also be added in the sum. As a result, by the guarantees of the exponential mechanism, we retrieve the optimal point with error at most

$$\alpha = \frac{6\mathrm{OPT}}{c_{\min} s\varepsilon'} \log d \log \frac{T}{\beta}, \quad \text{where } \varepsilon' = \frac{\varepsilon}{\sqrt{2T\log(1/\delta)}}.$$

Expanding, we get

$$s = O\left(\frac{\mathrm{OPT}^2 \log d \log^{1/2} m \log \frac{T}{\beta} \log^{1/2}(1/\delta)}{c^2 \alpha^2 \varepsilon}\right).$$

Appendix I. Full Proof of Theorem 3.3

Proof. Privacy. Let $\mathcal{G}$ be the event, over $\mathcal{H}$'s internal randomness, that $\mathcal{H}$ is a correct k-MIPS oracle for *every possible* query vector $q \in \mathbb{R}^{|\mathcal{X}|}$. By assumption, $\Pr[\mathcal{G}] \ge 1 - \frac{1}{m}$. Crucially, $\mathcal{G}$ is defined purely in terms of $\mathcal{H}$'s construction and holds for all query vectors simultaneously, so it is *independent of the database*. In particular, its probability does not depend on X, even though the actual queries submitted to $\mathcal{H}$ during an execution — which are of the form $h - p^{(t)}$ where h is the histogram of X — are data-dependent.

Let $X \sim X'$ be neighboring datasets, let $\mathcal{A}$ be Algorithm 2, and let E be any measurable event over the output space. We decompose by conditioning on $\mathcal{H}$:

$$\begin{aligned}
\Pr[\mathcal{A}(X) \in E] &= \mathop{\mathbb{E}}_{\mathcal{H}}[\Pr[\mathcal{A}(X) \in E \mid \mathcal{H}]] \\
&= \mathop{\mathbb{E}}_{\mathcal{H}}[\Pr[\mathcal{A}(X) \in E \mid \mathcal{H}] \cdot \mathbf{1}[\mathcal{G}(\mathcal{H})]] \\
&\quad + \mathop{\mathbb{E}}_{\mathcal{H}}\big[\Pr[\mathcal{A}(X) \in E \mid \mathcal{H}] \cdot \mathbf{1}[\overline{\mathcal{G}}(\mathcal{H})]\big] \\
&\le \mathop{\mathbb{E}}_{\mathcal{H}}[\Pr[\mathcal{A}(X) \in E \mid \mathcal{H}] \cdot \mathbf{1}[\mathcal{G}(\mathcal{H})]] + \Pr[\overline{\mathcal{G}}].
\end{aligned}$$

For any fixed realization of $\mathcal{H}$ for which $\mathcal{G}(\mathcal{H})$ holds, $\mathcal{H}$ returns the correct top-$\sqrt{m}$ set for *every* query vector, including the data-dependent queries $h - p^{(t)}$ submitted during execution. Therefore every call to LazyEM is distributionally identical to a call to the standard exponential mechanism EM. Since $\mathcal{H}$ is fixed, the remaining randomness of $\mathcal{A}$ consists only of the Gumbel samples and binomial draws, making the algorithm equivalent to running standard MWEM: T sequential, adaptive calls to a $(\varepsilon_0, 0)$-DP exponential mechanism. By Theorem C.1, this composition is (ε, δ)-DP. Therefore, for any fixed $\mathcal{H}$ with $\mathcal{G}(\mathcal{H})$:

$$\Pr[\mathcal{A}(X) \in E \mid \mathcal{H}] \le e^{\varepsilon} \cdot \Pr[\mathcal{A}(X') \in E \mid \mathcal{H}] + \delta.$$

Substituting back and using $\Pr[\overline{\mathcal{G}}] \le \frac{1}{m}$:

$$\Pr[\mathcal{A}(X) \in E] \le \mathop{\mathbb{E}}_{\mathcal{H}}\left[\left(e^{\varepsilon}\Pr[\mathcal{A}(X') \in E \mid \mathcal{H}] + \delta\right) \cdot \mathbf{1}[\mathcal{G}(\mathcal{H})]\right] + \frac{1}{m}$$
$$\le e^{\varepsilon} \cdot \Pr[\mathcal{A}(X') \in E] + \delta + \frac{1}{m},$$

establishing $\left(\varepsilon, \delta + \frac{1}{m}\right)$-DP.

Runtime. Each iteration of MWU takes $O(|\mathcal{X}|\sqrt{m})$ time in expectation, as claimed.

Utility. On the event $\mathcal{G}$, LazyEM is distributionally identical to EM, so the utility guarantees are identical to those of Theorem 3.1. Thus, if

$$n = \Omega\left(\frac{\log\frac{1}{\delta}\log\frac{m}{\beta}\sqrt{\log|\mathcal{X}|}}{\varepsilon\alpha^2}\right),$$

then with probability at least $1 - \beta - \frac{1}{m}$ we have $\|Q \cdot (\widehat{p} - h)\|_\infty \le \alpha$. □

Appendix J. Index Configuration

We compare three FAISS index types for approximate nearest neighbor search:

- The **Flat** index serves as the baseline, performing exact inner product search via linear scan with $O(m)$ complexity per query.
- For **IVF** (Inverted File Index), we partition the query vectors into $\texttt{nlist} = \max(2\sqrt{m}, 20)$ Voronoi cells using k-means clustering, and search the $\texttt{nprobe} = \min(\texttt{nlist}/4, 10)$ nearest cells at query time. This reduces the search space from m to approximately $m \cdot \texttt{nprobe}/\texttt{nlist}$ vectors on average.
- For **HNSW** (Hierarchical Navigable Small World), we construct a proximity graph with $M = 32$ neighbors per node, using $\texttt{efConstruction} = 100$ during index building and $\texttt{efSearch} = 64$ during querying. HNSW achieves approximately $O(\log m)$ query complexity by navigating through a hierarchical graph structure.

Both approximate indices trade off some accuracy for improved query time, with IVF offering faster index construction and HNSW providing better query performance at scale. All experiments were run on a MacBook Air with 16 GB of RAM on a single CPU core.

Appendix K. Additional Experiments on Linear Queries

In this section we provide additional experimental results and ablation studies for the application of Fast MWEM on the linear query release problem. Figures 1b and 1c (in the main paper) support the main discussion in Section 5; the experiments below use a flat index when the goal is accuracy rather than wall-clock performance.

K.1. **Analyzing the margin B.** Recall that in Fast MWEM, apart from the top $\sqrt{m}$ scores S we also need to sample C scores outside of S, where $C \sim \mathrm{Bin}(n - \sqrt{m}, 1 - \exp(-\exp(-B)))$. Here B is a margin we calculate based on the maximum pertrubed score and the minimum score in S. If B is very large, we risk having to collect a large additional sample, effectively losing the sublinear runtime guarantee.

Mussmann et al. [2017] show that in expectation we have $C = O(\sqrt{m})$. We ran an additional experiment to verify that this is the case. Over $T = 500$ iterations and for $m \in \{500, 2000, 20000\}$ we calculate C and the percentage of m that we need to take additional samples of. **We confirm**

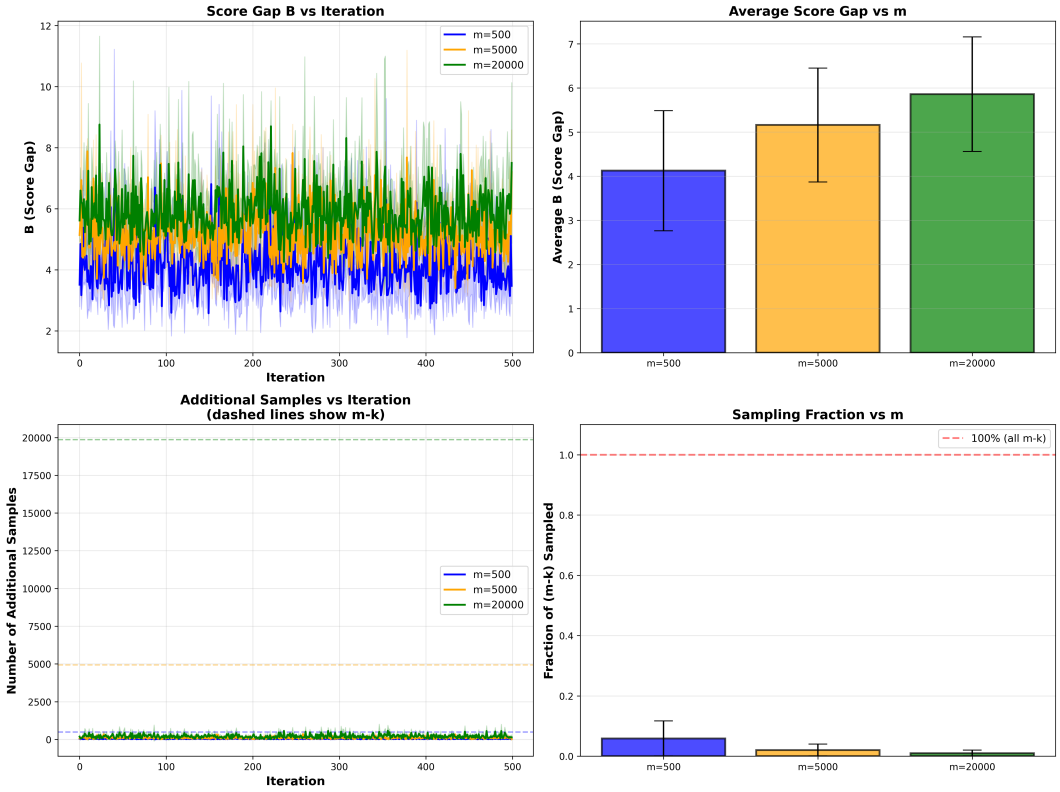


FIGURE 2. Analyzing the margin B

that the fraction of additional samples we take is a very small fraction of m, in the order of $O(\frac{1}{\sqrt{m}})$.

K.2. **Ablation with number of samples.** We also verify that the number of samples n is indeed related to the error of our mechanism, in the same way it is for MWEM. Setting $m = 100$ and letting $T = n^2$, **our experiment confirms the fact that more samples lead to increased accuracy**. Both MWEM and Fast MWEM behave very similarly in this experiment, which further confirms the closeness in the algorithms' behavior.

APPENDIX L. ADDITIONAL PLOTS ON LINEAR PROGRAMMING WITH FAST MWEM

Below we provide additional plots.

- As Figure 6 shows, both IVF and HNSW perform almost identical iterations as the original MWEM (No Index). Notably the performance of HNSW in total runtime is superior to the other approaches.
- When m is ablated with very large values ($3 \cdot 10^5 \leq m \leq 15 \times 10^5$), the runtime of HNSW manifests the sublinear speed-up by completely outperforming the other approaches, even though the build time of the data structure is larger. Unfortunately, IVF did not give us signficant speed up in these experiments.

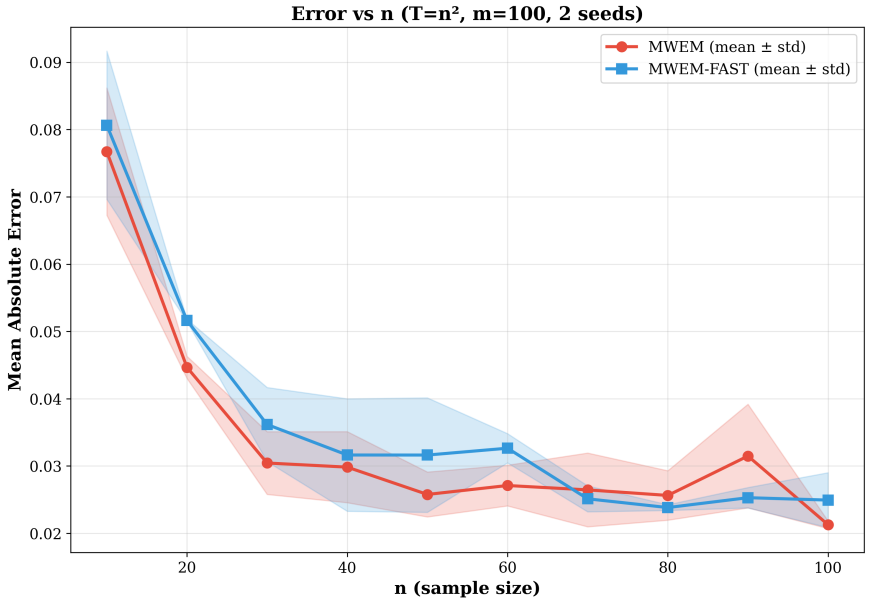
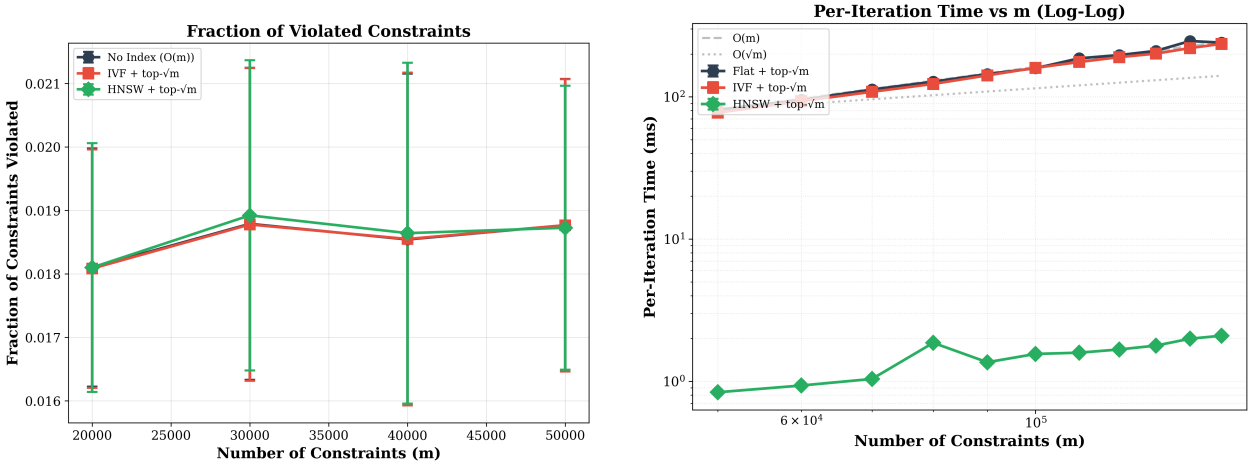


FIGURE 3. Ablation of the final error for different values of n



(A) Violated constraints across indices.

(B) Runtime vs. m across indices.

FIGURE 4. Scalar-private LP experiments ($d = 20$, $\Delta_\infty = 0.1$, $\alpha = 0.5$). HNSW achieves $O(\sqrt{m})$ runtime scaling with negligible accuracy loss.

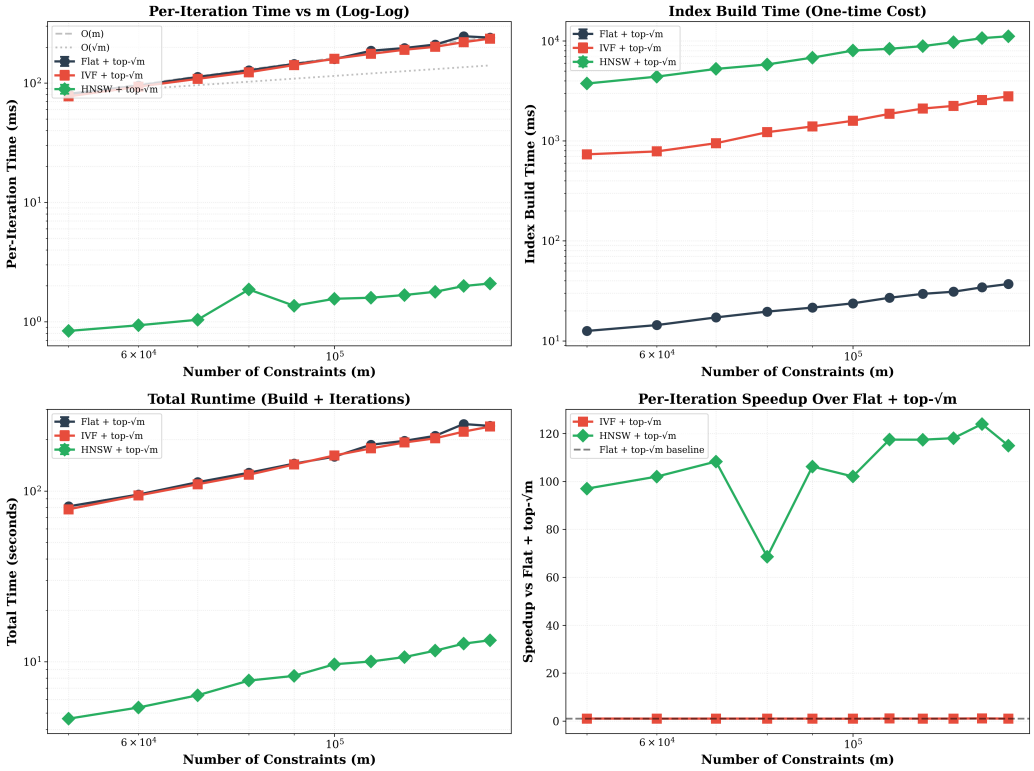


FIGURE 5. Comparing index performance for large ablated values of m

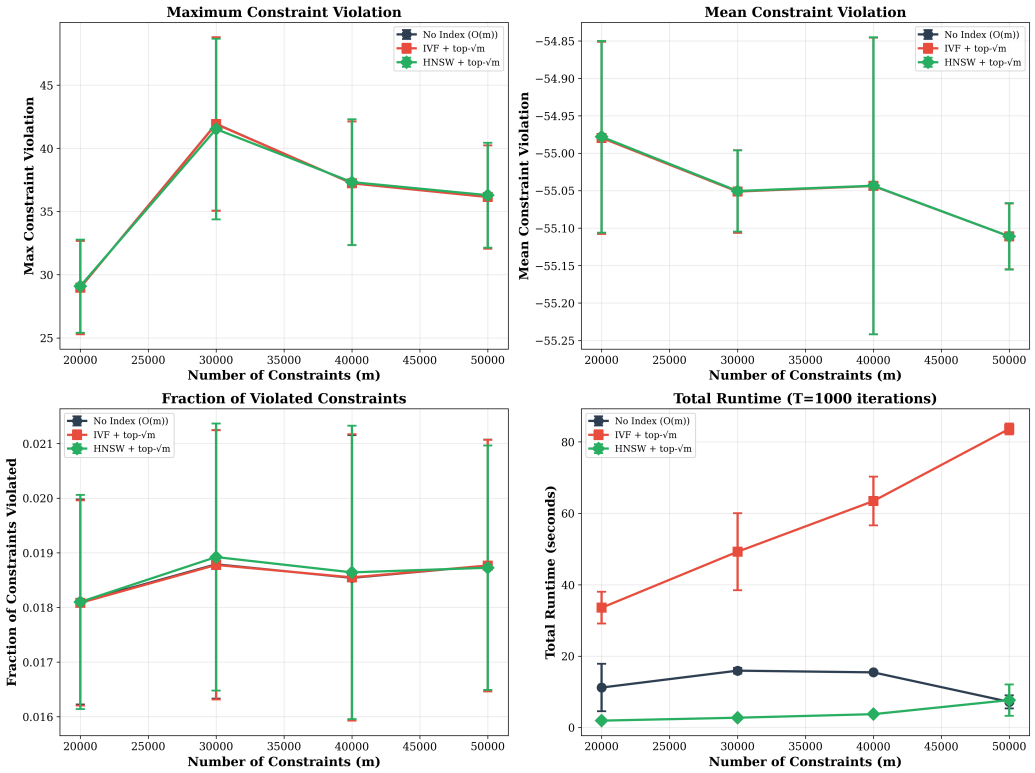


FIGURE 6. Comparing the error and max-constraint violations for solving scalar-private LPs with fast MWEM.