arXiv:2310.13174v2 [cs.DS] 21 Dec 2023

Faster Algorithms for Text-to-Pattern Hamming Distances*

Timothy M. Chan† UIUC
Ce Jin‡ MIT
Virginia Vassilevska Williams§ MIT
Yinzhan Xu¶ MIT

Abstract

We study the classic *Text-to-Pattern Hamming Distances* problem: given a pattern P of length m and a text T of length n, both over a polynomial-size alphabet, compute the Hamming distance between P and $T[i\,..\,i+m-1]$ for every shift i, under the standard Word-RAM model with $\Theta(\log n)$-bit words.

- We provide an $O(n\sqrt{m})$ time Las Vegas randomized algorithm for this problem, beating the decades-old $O(n\sqrt{m\log m})$ running time [Abrahamson, SICOMP 1987]. We also obtain a deterministic algorithm, with a slightly higher $O(n\sqrt{m}(\log m\log\log m)^{1/4})$ running time.
  Our randomized algorithm extends to the k-bounded setting, with running time $O\big(n+\frac{nk}{\sqrt{m}}\big)$, removing all the extra logarithmic factors from earlier algorithms [Gawrychowski and Uznański, ICALP 2018; Chan, Golan, Kociumaka, Kopelowitz and Porat, STOC 2020].
- For the $(1+\varepsilon)$-approximate version of Text-to-Pattern Hamming Distances, we give an $\widetilde{O}(\varepsilon^{-0.93}n)$ time Monte Carlo randomized algorithm (where $\widetilde{O}$ hides poly-logarithmic factors), beating the previous $\widetilde{O}(\varepsilon^{-1}n)$ running time [Kopelowitz and Porat, FOCS 2015; Kopelowitz and Porat, SOSA 2018].

Our approximation algorithm exploits a connection with 3SUM, and uses a combination of Fredman's trick, equality matrix product, and random sampling; in particular, we obtain new results on approximate counting versions of 3SUM and Exact Triangle, which may be of independent interest. Our exact algorithms use a novel combination of hashing, bit-packed FFT, and recursion; in particular, we obtain a faster algorithm for computing the sumset of two integer sets, in the regime when the universe size is close to quadratic in the number of elements.

We also prove a fine-grained equivalence between the exact Text-to-Pattern Hamming Distances problem and a range-restricted, counting version of 3SUM.

*Timothy Chan is partially supported by NSF grant CCF-2224271. Ce Jin is partially supported by NSF grants CCF-2129139 and CCF-2127597. Virginia Vassilevska Williams and Yinzhan Xu are partially supported by NSF grants CCF-2129139 and CCF-2330048 and BSF Grant 2020356.

†tmc@illinois.edu
‡cejin@mit.edu
§virgi@mit.edu
¶xyzhan@mit.edu

1 Introduction

In this paper, we study one of the most basic problems about string matching, the classic *Text-to-Pattern Hamming Distances* problem (also known as Sliding Window Hamming Distances, or String Matching with Mismatches): given a pattern P of length m and a text T of length n over an alphabet of size σ, compute the Hamming distance (i.e., the number of mismatches) between P and $T[i\,..\,i+m-1]$ for every shift i.

Fischer and Paterson's seminal work [FP74] gave an algorithm running in $O(\sigma n \log m)$ time[1] by reducing it to convolution or polynomial multiplication, which can be solved using the Fast Fourier Transform (FFT); this is the fastest known algorithm for small σ. For arbitrary alphabet size, well-known work by Abrahamson [Abr87] described an $O(n\sqrt{m}\,\mathrm{polylog}\,n)$ time algorithm for a family of generalized string matching problems; for Text-to-Pattern Hamming Distances, the time bound is $O(n\sqrt{m\log m})$. Abrahamson's algorithm was perhaps the first example of a string algorithm with "intermediate complexity" between linear and quadratic (ignoring logs). A fine-grained reduction attributed to Indyk (see [Cli09]) shows that no combinatorial algorithm for Text-to-Pattern Hamming Distances can run in $O(nm^{1/2-\delta})$ time for an arbitrarily small constant $\delta > 0$, under the combinatorial Boolean Matrix Multiplication Hypothesis.[2] This suggests that Abrahamson's algorithm might be optimal up to sub-polynomial factors, at least for combinatorial algorithms.

However, so far not even poly-logarithmic improvements of Abrahamson's algorithm have been reported. This is not due to a lack of interest. In fact, many algorithms are designed to shave logarithmic factors for stringology problems (e.g. [CGK+20, MP80, Mye92, Ind98, CLZ03, BF08, BT09, Gra16]). In this paper, we will tackle the following decades-old question:

Open Question 1. *Can we improve Abrahamson's $O(n\sqrt{m\log m})$ time algorithm for Text-to-Pattern Hamming Distances?*

We remark that Fredriksson and Grabowski [FG13] designed a faster algorithm for Text-to-Pattern Hamming Distances when the word size w is $\omega(\log n)$ and $m = O(\frac{n\log m}{w})$. However, their algorithm is not faster in the common Word-RAM model with $w = \Theta(\log n)$, which is the model we consider here.

To obtain faster algorithms for the Text-to-Pattern Hamming Distances problem, researchers have considered two easier versions: the $(1+\varepsilon)$-approximate version and the k-bounded version. Next we summarize previous results in these two settings.

$(1+\varepsilon)$-approximation. The $(1+\varepsilon)$-approximate version asks to approximate the Hamming distance between P and $T[i\,..\,i+m-1]$ for every shift i within a $(1+\varepsilon)$ factor of the true distance, for $\varepsilon > 0$.

In 1993, Karloff [Kar93] gave a randomized (Monte Carlo) algorithm running in $O(\varepsilon^{-2} n \log n \log m)$ time with high success probability.[3] Karloff also derandomized his algorithm at the cost of only an extra logarithmic factor.

Karloff's $\widetilde{O}(\varepsilon^{-2}n)$ time algorithm remained the state-of-the-art for a long time (and unimproved except in some special cases [AGW13]).[4] This ε^{-2} dependency is due to the variance that results from random projections, and it was thought to be inherent as suggested by the $\Omega(\varepsilon^{-2})$ lower bound for computing

[1] We consider the Word-RAM model with $\Theta(\log n)$-bit words throughout the paper.

[2] While the notion of "combinatorial" is not well-defined, the typical notion of a combinatorial algorithm is one that does not use fast matrix multiplication. The combinatorial Boolean Matrix Multiplication hypothesis states that there is no combinatorial algorithm that multiplies two $n \times n$ Boolean matrices in $O(n^{3-\varepsilon})$ time, for any constant $\varepsilon > 0$, in the word-RAM model with $\Theta(\log n)$ bit words.

[3] With high probability (w.h.p.) means probability $1 - O(n^{-c})$ for arbitrarily large constant c.

[4] We use $\widetilde{O}$ to hide poly-logarithmic factors in the input size.

Hamming distance in the one-way communication model [Woo04, JKS08]. Hence, it came as a surprise when Kopelowitz and Porat in STOC'15 [KP15] gave a faster algorithm in $\widetilde{O}(\varepsilon^{-1}n)$ time, using techniques from sparse recovery. This algorithm was subsequently simplified (and improved in terms of logarithmic factors) by Kopelowitz and Porat [KP18], with a time bound of $O(\varepsilon^{-1}n\log n\log m)$ (with high success probability). See Table 2. It is unclear whether this ε^{-1} dependency is best possible, and this leads to the following tantalizing question:

Open Question 2. *Can we improve Kopelowitz and Porat's $\widetilde{O}(\varepsilon^{-1}n)$ time algorithm for $(1+\varepsilon)$-approximate Text-to-Pattern Hamming Distances?*

Generally, there has been growing interest in understanding the ε-dependencies needed to solve fundamental problems in fine-grained complexity (e.g., partition [MWW19, BN21b, WC22] and knapsack [Cha18, Jin19, BC22, DJM23, Mao23, CLMZ23]). Such ε-dependencies are especially important when one demands very accurate answers (e.g., computing $(1+\frac{1}{\sqrt{m}})$-approximations).

More recently, Chan, Golan, Kociumaka, Kopelowitz and Porat in STOC'20 [CGK+20] partially answered Open Question 2: when the pattern length m satisfies $m \geq \varepsilon^{-28}$, one can $(1+\varepsilon)$-approximate Text-to-Pattern Hamming Distances in $\widetilde{O}(n)$ time, without any $\varepsilon^{-O(1)}$ factors. The assumption may be relaxed to $m \geq \varepsilon^{-10}$ if the matrix multiplication exponent ω is equal to 2, and if the goal is to obtain better than $\widetilde{O}(\varepsilon^{-1}n)$ time instead of $\widetilde{O}(n)$, the assumption can be relaxed further by re-analyzing/modifying their algorithm. However, inherently their approach is unable to beat $\varepsilon^{-1}n$ if ε^{-1} is large, for example, when ε^{-1} is $m^{1/3}$ or $\sqrt{m}$. The $\varepsilon^{-1}=\sqrt{m}$ case is particularly instructive: here, $\widetilde{O}(\varepsilon^{-1}n)$ coincides with the $\widetilde{O}(n\sqrt{m})$ bound for the exact problem; for distances that are $\Theta(m)$, we are demanding $O(\sqrt{m})$ additive error, and sampling-based approaches do not seem to offer any speedup (if we try to estimate distances by sampling different positions of the pattern string, we would need a sample size of $\Omega(m)$, which is not any smaller than the length of the original pattern string).

Chan et al. [CGK+20] also gave an $O(\varepsilon^{-2}n)$ time randomized algorithm (correct with high probability) without any logarithmic factors, which is preferable when ε^{-1} is small. Both Open Question 1 and Open Question 2 were explicitly asked during a talk on [CGK+20] given by Kociumaka.[5]

Other variations of $(1+\varepsilon)$-approximation text-to-pattern matching problem have also been studied in the literature, such as replacing Hamming distance by other ℓ_p norms [LP11, GU18, SU19, Uzn20a], or restricting to algorithms that do not use FFT [CGK+20, Uzn20a]. See also the survey by Uznański [Uzn20b].

k-mismatch. Given a threshold k, the k-bounded (or k-mismatch) version of Text-to-Pattern Hamming Distances asks to compute the Hamming distances only for locations with distances at most k, and output ∞ for other locations.

After a long line of works [LV86, LV89, GG86, SV96, CH02, ALP04, CFP+16, GU18, CGK+20] (see Table 1), the current fastest algorithm by Chan, Golan, Kociumaka, Kopelowitz and Porat [CGK+20] is a Monte Carlo randomized algorithm (correct with high probability) in $O\big(n+\min\big(\frac{nk}{\sqrt{m}}\sqrt{\log m},\frac{nk^2}{m}\big)\big)$ time, shaving off some logarithmic factors from the earlier deterministic algorithm by Gawrychowski and Uznański [GU18] in $O\big(n\log^2 m\log\sigma+\frac{nk\sqrt{\log n}}{\sqrt{m}}\big)$ time. Gawrychowski and Uznański [GU18] also extended Indyk's fine-grained reduction (mentioned in the notes of Clifford [Cli09]) to show a tight conditional lower bound for combinatorial algorithms solving the k-mismatch problem under the Boolean Matrix Multiplication Hypothesis.

[5] Talk video link: https://youtu.be/WEiQjjTBX-4?t=2820

1.1 Our results

In this paper, we give new exact and approximation algorithms for Text-to-Pattern Hamming Distances, answering both Open Question 1 and Open Question 2 in the affirmative.

Theorem 1.1 (Approximation algorithm with sublinear $1/\varepsilon$ dependence)**.** *The $(1+\varepsilon)$-approximate Text-to-Pattern Hamming Distances problem can be solved by a Monte Carlo randomized algorithm in $\widetilde{O}(\varepsilon^{-\gamma}n)$ time, where $\gamma = 8/(11-\omega) < 0.928$.*

Here, $\omega \in [2, 2.372)$ is the matrix multiplication exponent [DWZ22, VXXZ23]. Our result resolves Open Question 2, showing that $\widetilde{O}(\varepsilon^{-1}n)$ [KP15, KP18] is not the ultimate answer for this problem (and, in particular, that it is possible to obtain polynomial improvements over $\widetilde{O}(n\sqrt{m})$ in the critical case of $\varepsilon^{-1} = \sqrt{m}$).

Theorem 1.2 (Exact algorithm without log factors)**.** *The Text-to-Pattern Hamming Distances problem can be solved by a Las Vegas algorithm which terminates in $O(n\sqrt{m})$ time with high probability.*

This result is the first speedup over Abrahamson's algorithm [Abr87] for more than three decades. We also give a new deterministic algorithm that runs faster than Abrahamson's algorithm, but slower than Theorem 1.2.

Theorem 1.3 (Deterministic exact algorithm)**.** *The Text-to-Pattern Hamming Distances problem can be solved by a deterministic algorithm in $O(n\sqrt{m}(\log m \log\log m)^{1/4})$ time.*

Our exact algorithms actually apply to the harder problem of computing $|\{j : P[j] \le T[i+j]\}|$ for all shifts i. (Note that the Text-to-Pattern Hamming Distances problem reduces to two instances of this problem, one of which with an alphabet in reversed order.) This is also known as the *Dominance Convolution* problem (see e.g., [AF91, LUW19]).

Theorem 1.4 (Exact algorithm for Text-to-Pattern Dominance Matching)**.** *The Text-to-Pattern Dominance Matching problem can be solved by a Las Vegas algorithm which terminates in $O(n\sqrt{m})$ time with high probability, or a deterministic algorithm which terminates in $O(n\sqrt{m}(\log m \log\log m)^{1/4})$ time.*

[LUW19] also observed that this Text-to-Pattern Dominance Matching problem is equivalent to the following "threshold" problem [AD11]: for a fixed δ, compute $|\{j : |P[j] - T[i+j]| > \delta\}|$ for all shifts i. Hence, this threshold problem can also be solved in the same time complexity as in Theorem 1.4.

Our technique also yields improvement to the k-mismatch problem.

Theorem 1.5 (k-mismatch algorithm without log factors)**.** *The k-bounded Text-to-Pattern Hamming Distances problem can be solved by a Monte Carlo algorithm in $O\big(n + \frac{nk}{\sqrt{m}}\big)$ expected time which outputs correct answers with high probability.*

This speeds up the previous $O\big(n + \min\big(\frac{nk}{\sqrt{m}}\sqrt{\log m}, \frac{nk^2}{m}\big)\big)$-time Monte Carlo algorithm (with high success probability) time by Chan, Golan, Kociumaka, Kopelowitz and Porat [CGK+20], and cleans up *all* the extra factors from the long line of previous works shown in Table 1 (note that $n + \frac{nk^2}{m}$ is never better than $n + \frac{nk}{\sqrt{m}}$).

Finally, we consider the fine-grained complexity of Text-to-Pattern Hamming Distances. As mentioned earlier, a reduction by Indyk (see [Cli09]) gives a tight conditional lower bound for combinatorial Text-to-Pattern Hamming Distances algorithms under the Boolean Matrix Multiplication Hypothesis. Indyk's reduction only gives an $n \cdot m^{\omega/2-1-o(1)}$ conditional lower bound for arbitrary (potentially non-combinatorial) algorithms. This lower bound is only non-trivial if $\omega > 2$.

In the Appendix we observe[6] that Indyk's reduction can be easily extended to start from the Equality Product of matrices [Vas15], which is known to be equivalent to Dominance Product [VW09, Mat91, LUW19, Vas15]). Equality Product and Dominance Product are among the so called "intermediate" matrix products [LPW20] believed to require $n^{2.5-o(1)}$ time, even if $\omega = 2$ (see also [LUW19]). The observation gives a higher, $nm^{1/4-o(1)}$ time fine-grained lower bound for Text-to-Pattern Hamming Distances against potentially non-combinatorial algorithms which holds even if $\omega = 2$. Similarly, Gawrychowski and Uznański's reduction [GU18] from Matrix Multiplication to the k-mismatch problem can also be extended this way, giving a higher $\frac{n^{1-o(1)}k}{m^{3/4}}$ fine-grained lower bound against potentially non-combinatorial algorithms which holds even if $\omega = 2$ and is only off by an $m^{1/4}$ factor from the known combinatorial algorithms for the problem.

Finally, we examine the relationship between Text-to-Pattern Hamming Distances and the well-studied 3SUM problem. It has long been asked (see e.g. [Uzn20b]) whether one can reduce 3SUM to Text-to-Pattern Hamming Distances.

Recently, Chan, Vassilevska Williams and Xu [CVX23] showed that 3SUM is *fine-grained equivalent* to the following counting version called #All-Nums-3SUM.

Problem 1 (#All-Nums-3SUM)**.** *Given three size N sets A, B, C of integers, for every $c \in C$, compute the number of $(a, b) \in A \times B$ where $a + b = c$.*

We consider the following variant of #All-Nums-3SUM in which one of the input sets is assumed to contain integers from a small range (3SUM where the numbers of one of the three sets are from a small range was mentioned in [CL15]).

Problem 2. *Given three size N sets A, B, C where $C = [N]$, for every $c \in C$, compute the number of $(a, b) \in A \times B$ where $a + b = c$.*

We show that Text-to-Pattern Hamming Distances when $n = O(m)$ is *equivalent* to Problem 2. This at least partially addresses the relationship between Text-to-Pattern Hamming Distances and 3SUM, as Problem 2 can be viewed as a range-restricted version of 3SUM (as 3SUM is equivalent to #All-Nums-3SUM).

Theorem 1.6 (Equivalence with a variant of 3SUM)**.** *If Problem 2 has a $f(N)$ time algorithm, then Text-to-Pattern Hamming Distances with $n = O(m)$ has an $\widetilde{O}(f(m))$ time algorithm, and vice versa.*

Bringmann and Nakos [BN20] designed a reduction from Text-to-Pattern Hamming Distances to a problem called Interval-Restricted Convolution, which is more general than Problem 2, and their reduction also works from Text-to-Pattern Hamming Distances to Problem 2. We show that the reduction is also possible in the other direction from Problem 2 to Text-to-Pattern Hamming Distances, establishing the equivalence.

1.2 Technical overview

Both our approximation and exact algorithms for Text-to-Pattern Hamming Distances use interesting new techniques, on which we now briefly elaborate.

Approximation algorithm. Our new approximation algorithm for Theorem 1.1 uses an approach that is markedly different from all previous approximation algorithms. The algorithms by Karloff [Kar93] and Kopelowitz and Porat [KP18] used random projection to reduce the alphabet size; afterwards, the problem

[6]The authors thank Amir Abboud and Arturs Backurs for discussions around this observation in 2015–2016.

reference	run time
Fischer and Paterson [FP74]	$O(\sigma n \log m)$
Abrahamson [Abr87]	$O(n\sqrt{m \log m})$
new	$O(n\sqrt{m})$
Landau and Vishkin [LV86, LV89] / Galil and Giancarlo [GG86]	$O(nk)$
Sahinalp and Vishkin [SV96]	$O(n + \frac{nk^{O(1)}}{m})$
Cole and Hariharan [CH02]	$O(n + \frac{nk^4}{m})$
Amir, Lewenstein and Porat [ALP04]	$O(\min\{n\sqrt{k \log k},\ n \log k + \frac{nk^3 \log k}{m}\})$
Clifford, Fontaine, Porat, Sach, and Starikovskaya [CFP+16]	$O(n \log^{O(1)} m + \frac{nk^2 \log k}{m})$
Gawrychowski and Uznański [GU18]	$O(n \log^2 m \log \sigma + \frac{nk\sqrt{\log n}}{\sqrt{m}})$
Chan, Golan, Kociumaka, Kopelowitz and Porat [CGK+20]	$O(n + \min\{\frac{nk\sqrt{\log m}}{\sqrt{m}},\ \frac{nk^2}{m}\})$
new	$O(n + \frac{nk}{\sqrt{m}})$

Table 1: Exact algorithms for the text-to-pattern Hamming distance problem (randomization allowed).

reference	run time	techniques
Karloff [Kar93]	$\widetilde{O}(\varepsilon^{-2}n)$	random projection
Indyk [Ind98]	$\widetilde{O}(\varepsilon^{-3}n)$	random sampling
Kopelowitz and Porat [KP15]	$\widetilde{O}(\varepsilon^{-1}n)$	projection with sparse recovery
Kopelowitz and Porat [KP18]	$\widetilde{O}(\varepsilon^{-1}n)$	random projection
Chan, Golan, Kociumaka, Kopelowitz and Porat [CGK+20]	$\widetilde{O}(n)$ for $m \gg \varepsilon^{-28}$	random sampling + rect. matrix mult.
new	$\widetilde{O}(\varepsilon^{-0.93}n)$	#3SUM techniques with matrix mult.

Table 2: Approximation algorithms for the text-to-pattern Hamming distance problem, focusing on ε-dependencies and ignoring logarithmic factors (randomization allowed).

can be solved by FFT. Karloff's algorithm required $O(\varepsilon^{-2})$ projections, whereas Kopelowitz and Porat's algorithm required a reduced alphabet size of $O(\varepsilon^{-1})$. On the other hand, the algorithms by Indyk [Ind98] and Chan et al. [CGK+20] used random sampling to examine selected positions of the pattern and text strings. An application of the Chernoff bound leads to $O(\varepsilon^{-2})$ factors in the running time, which are too big for the critical case $\varepsilon^{-1} = \sqrt{m}$.

In contrast, our new algorithm follows an approach that is actually closer to the known exact algorithms. We view the problem as a certain colored counting variant of 3SUM (where colors correspond to characters in the alphabet), which can be decomposed into multiple instances of an uncolored counting 3SUM problem (one per character in the alphabet).

Recently, Chan, Vassilevska Williams and Xu [CVX23] gave reductions from counting versions of basic problems in fine-grained complexity, including 3SUM and Exact-Triangle (finding triangles with weight exactly zero in a dense weighted graph), to their original versions. They obtained their results using a simple combination of "Fredman's trick"[7] and Equality Product. We show that their ideas, originally developed for proving fine-grained equivalences and conditional lower bounds, can be adapted to design faster algorithms

[7]The (trivial) observation that $a + b \leq a' + b'$ is equivalent to $a - a' \leq b' - b$, which is the key insight behind Fredman's seminal paper on all-pairs shortest paths (APSP) [Fre76], and used in many subsequent works on APSP and 3SUM (e.g., [Tak98, Cha10, Wil18, GP18, Cha20]).

for *approximate* counting versions of 3SUM and Exact-Triangle.

More specifically, Fredman's trick and Equality Product allow us to compute counts in the case when counts are large (i.e., when the number of "witnesses" is large) [CVX23]. Chan, Vassilevska Williams and Xu then used oracles to handle the case when counts are small (since they were designing reductions, not algorithms), but we observe that the small-count case is actually easier in the context of approximation: we can use random sampling with smaller sample sizes, since the variance is lower.

To summarize, our new algorithm is technically interesting for multiple reasons:

1. It further illustrates the power of the "Fredman's trick meets Equality Product" technique from [CVX23], in the context of approximation algorithms. These ideas might spawn further applications.

2. It demonstrates that fine-grained reductions (originally developed for proving conditional lower bounds) can help in the design of algorithms. In particular, our algorithm makes essential use of the known chain of reductions from Convolution-3SUM to Exact-Triangle [VW09], and from 3SUM to Convolution-3SUM [CH20b].

3. Its use of matrix multiplication is non-trivial and interesting. It is open whether fast matrix multiplication helps for the exact Text-to-Pattern Hamming Distances problem, but our new algorithm demonstrates that it helps for the approximate problem. Chan et al.'s previous algorithm [CGK+20] also used rectangular matrix multiplication to speed up certain steps, but our algorithm here relies on matrix multiplication (via Equality Product) in a more essential way.

Exact algorithms. Our exact algorithm for Theorem 1.2 also works by decomposing into multiple 3SUM-like subproblems (one per character of the alphabet). More precisely, a subproblem corresponds to computing a sumset $X + Y$ (where we also want the counts/multiplicities per element) for two given sets of n integers in $[U]$; equivalently, this corresponds to computing the convolution of two sparse binary vectors in $[U]$ with n nonzero entries. The critical case in our application turns out to be when U is below and close to n^2; this is when the standard $O(U \log U)$-time algorithm by FFT does not outperform the brute-force $O(n^2)$-time algorithm. We present a new lemma showing that the sumset/convolution can be computed in $O(U \log(n^2/U))$ expected time, which outperforms both FFT and brute-force, and is good enough to shave off all the extra logarithmic factors and yield the $O(n\sqrt{m})$ bound for the exact Text-to-Pattern Hamming Distances problem.

A number of algorithms have already been developed for sparse convolution [CH02, CL15, BFN22]. The current fastest algorithm by Bringmann et al. [BFN22] is complicated, and does not address our question of speeding up $O(U \log U)$ in the regime of U close to n^2. Our new algorithm shares some ideas from these previous algorithms, but is arguably simpler, and more accessible, than the one of [BFN22]. It only requires Dietzfelbinger's standard family of almost linear hash functions [Die96] (though we need to establish some new properties). Hashing is used to iteratively shrink the "support" (the subset of elements whose counts are not known yet), but the key twist is an extra use of recursion to identify candidates for "light" elements (elements with small counts). A bit-packed version of FFT is used to compute counts with a small modulus.

By further incorporating some ideas from [CL15], we obtain a derandomization (Theorem 1.3), albeit with an extra factor of about $\log^{1/4} m$.

It is straightforward to combine our new lemma with existing algorithms [GU18, CGK+20] to obtain our result on the k-mismatch problem (Theorem 1.5) and the Dominance Convolution problem (Theorem 1.4).

2 Preliminaries

A string S of length $|S| = s$ is a sequence of characters $S[1]S[2]\dots S[s]$ over an alphabet Σ. We assume the alphabet has size $|\Sigma| = \sigma \leq n^{O(1)}$, and identify Σ with the set of integers in $[\sigma]$. For $1 \leq i \leq j \leq s$, we denote the substring $S[i]S[i+1]\dots S[j]$ of S by $S[i\,..\,j]$.

We use $[n]$ to denote $\{0, 1, \dots, n-1\}$.

Definition 2.1. Given two length-n sequences $\langle a_0, \dots, a_{n-1}\rangle$ and $\langle b_0, \dots, b_{n-1}\rangle$, their convolution $c = a \star b$ is a length $2n-1$ sequence, where $c_i = \sum_{j=0}^{i} a_i b_{j-i}$ (assume that out-of-range array entries are set to 0).

It is well-known that we can compute the convolution between two integer sequences in $O(n \log n)$ time using FFT. If one instead only needs to compute the entries of c modulo some given prime p, then a slightly faster running time is possible, in the word-RAM model with $O(\log n)$ bit words:

Lemma 2.2. *Given a prime $p \leq n^{O(1)}$ and two length-n sequences a, b with entries in $\mathbb{F}_p$, we can deterministically compute $a \star b$ in $O(n \log p)$ time.*

Indyk [Ind98] claimed a proof of Lemma 2.2 for the case of $p = 2$, but his proof was incomplete. For the $p = 2$ case, his argument can be completed via a more recent work [LAHC16], but it does not extend to larger p. In Appendix B we point out the issue in Indyk's argument (and mention subsequent works affected by this issue), and then include a complete proof of Lemma 2.2 for general p that fixes this issue.

We use $M(n_1, n_2, n_3)$ to denote the time for computing the product between an $n_1 \times n_2$ matrix and an $n_2 \times n_3$ matrix. We use the following algorithm for computing the equality product between two matrices, which follows from known techniques [Mat91, Yus09] (see also [CVX23]).

Lemma 2.3. *Given an $n_1 \times n_2$ matrix A and an $n_2 \times n_3$ matrix B, their* equality product

$$C[i,j] := |\{k \in [n_2] : A[i,k] = B[k,j]\}|$$

can be computed in time

$$M_{\text{eq}}(n_1, n_2, n_3) \;=\; O\left(\min_{1 \leq r \leq n_2}\left(\frac{n_1 n_2 n_3}{r} + M(n_1, rn_2, n_3)\right)\right).$$

For example, in the case of square matrices, the above implies $M_{\text{eq}}(n,n,n) = O(\min_r(n^3/r + M(n, rn, n))) \leq O(\min_r(n^3/r + rn^\omega)) = O(n^{(3+\omega)/2})$.

3 Approximate Text-to-Pattern Hamming Distances

In this section, we begin by solving approximate counting variants of several core problems in fine-grained complexity—namely, Exact Triangle, Convolution-3SUM, and 3SUM. All this will then lead to an algorithm for approximate Text-to-Pattern Hamming Distances.

3.1 Approximate Counting All-Edges Exact Triangle

In recent work [CVX23], Chan, Vassilevska Williams and Xu proved fine-grained equivalences between several central fine-grained problems and their counting versions. Let us take the All-Edges Exact Triangles (AE-Exact-Triangle) problem for an example. In this problem, when given a weighted tripartite graph,

and for each edge, we need to decide whether this edge is in a triangle whose edge weights sum up to 0. In the counting version of AE-Exact-Triangle, we need to count the number of zero-weight triangles each edge is in. Equivalently, given 3 matrices A, B, and C, we need to count the number of ks such that $A[i,k] + B[k,j] = -C[i,j]$, for each i, j. Prior to [CVX23], via the technique in [VW18], it was known that AE-Exact-Triangle is equivalent to its counting version where all the per-edge counts are small. The key observation in [CVX23] is that, when the per-edge counts are big, we can efficiently compute the counts exactly, using Fredman's trick [Fre76] in combination with Equality Product.

The following lemma adapts this approach to an approximate counting setting with additive error. The main new idea is that when counts are small, we can use random sampling at a lower rate to estimate such counts, since the variance is lower. In fact, even in the case when counts are big, we can also use sampling at different rates to approximate the Equality Products a little more quickly.

Let us briefly discuss the lemma statement below. First, in the approximate setting of AE-Exact-Triangle, it turns out that the third matrix C is unnecessary: one can design a truly subcubic time algorithm that solves the problem for all C at the same time. Intuitively, the reason is that when additive error is allowed, small counts in principle may be approximated by zeros, and zero values need not be output explicitly. Thus, it suffices to estimate the count values $\text{COUNT}[i,j,z] := |\{k \in [n_2] : A[i,k] + B[k,j] = z\}|$ for the "heavy hitters" z with sufficiently large counts, but the number of such z is sublinear per (i,j).

Second, the lemma statement below bounds the variance of the estimators, instead of the additive error (which is bounded by the square root of the variance with good probability); this will be important, as we will later need to sum several estimators (if they are independent or uncorrelated, we can sum their variances).

Third, the lemma involves multiple parameters, and on first reading, it may be helpful to focus, throughout this section, on the simplest setting with $t = 1$ and $\Delta = 1$ (where one can ignore the condition about uncorrelation), which is already sufficient to address the critical case of the Hamming Distances problem when $\varepsilon^{-1} = \sqrt{m}$ and distances are $\Theta(m)$ (where we want $O(\sqrt{m})$ additive error).

Lemma 3.1. *Given an $n_1 \times n_2$ integer matrix A and an $n_2 \times n_3$ integer matrix B, where all values of A are divisible by positive integer $\Delta \le n_3$, define*

$$\text{COUNT}[i,j,z] := |\{k \in [n_2] : A[i,k] + B[k,j] = z\}|.$$

Given parameter $1 \le t \le n_2$, there is a randomized algorithm that computes estimates $f[i,j,z]$ over all $i \in [n_1]$, $j \in [n_3]$, and z, such that the expectation of $f[i,j,z]$ is equal to $\text{COUNT}[i,j,z]$, and the variance of $f[i,j,z]$ is $O(tn_2)$. (Zero entries of f need not be output explicitly.) Furthermore, $f[i,j,z]$ and $f[i',j',z']$ are uncorrelated if $(i,j) \neq (i',j')$ and $z \not\equiv z' \pmod{\Delta}$. The running time is

$$\widetilde{O}\left(\min_{1 \le s \le n_2/t}\left(\frac{n_1 n_2 n_3}{st} + s\Delta M(n_1, n_2/t, n_3/\Delta)\right)\right).$$

Proof. Define the *witness set* $W[i,j,z] = \{k \in [n_2] : A[i,k] + B[k,j] = z\}$.

- **Few-witnesses case.** Independently for each (i,j), pick a random subset $R_{ij} \subseteq [n_2]$ where each element in $[n_2]$ is put in R with probability $\rho_* := 1/(st)$. Define $f_*[i,j,z] = (1/\rho_*) \cdot |\{k \in R_{ij} : A[i,k]+B[k,j] = z\}|$. We can generate all nonzero values $f_*[i,j,z]$ by looping through each $i \in [n_1]$, $j \in [n_3]$, and $k \in R_{ij}$, in total time $\widetilde{O}(n_1 n_3 \cdot \rho_* n_2)$. (It is essential that we are not required to output zero values, since otherwise we would not be able to obtain $o(n_1 n_2 n_3)$ time.) Then $\mathbb{E}[f_*[i,j,z]] = \text{COUNT}[i,j,z]$, and $\text{Var}[f_*[i,j,z]] = (1/\rho_*^2) \cdot \text{COUNT}[i,j,z] \cdot (\rho_* - \rho_*^2) \le st \cdot \text{COUNT}[i,j,z]$, which is good when $\text{COUNT}[i,j,z]$ is small. As we pick R_{ij} independently for each (i,j), the estimates $f_*[i,j,z]$ and $f_*[i',j',z']$ are independent if $(i,j) \neq (i',j')$.

- **Many-witnesses case.** For $\ell = 0, \ldots, \log s$ (w.l.o.g., we assume s to be a power of 2), do the following:

  Pick a random subset $H^{(\ell)} \subseteq [n_2]$ of size $c2^\ell \log(n_1 n_2 n_3)$ for a sufficiently large constant c. Then $H^{(\ell)}$ hits $W[i,j,z]$ w.h.p. for each i,j,z with $\text{COUNT}[i,j,z] \geq n_2/2^\ell$.

  For each $k_0 \in H^{(\ell)}$ and $\xi \in [\Delta]$, pick a random subset $R^{(\ell,k_0,\xi)} \subseteq [n_2]$ where each element in $[n_2]$ is put in $R^{(\ell,k_0,\xi)}$ with probability $\rho_\ell := 1/(2^\ell t)$. For each $k_0 \in H^{(\ell)}$ and $\xi \in [\Delta]$, compute the equality product $C^{(k_0,\ell)}[i,j] = |\{k \in R^{(\ell,k_0,\xi)} : A[i,k] - A[i,k_0] = B[k_0,j] - B[k,j]\}|$ for $i \in [n_1]$ and $j \in [n_3]$ with $B[k_0,j] \bmod \Delta = \xi$. This takes total time $\widetilde{O}(2^\ell \Delta M_{\text{eq}}(n_1, \rho_\ell n_2, n_3/\Delta))$, by splitting each set $\{j \in [n_3] : B[k_0,j] \bmod \Delta = \xi\}$ into subsets of size $O(n_3/\Delta)$.

  Consider i,j,z such that $W[i,j,z]$ is hit by $H^{(\ell)}$, i.e., $z = A[i,k_0] + B[k_0,j]$ for some $k_0 \in H^{(\ell)}$. Let k_0 be the smallest such index. Define $f_\ell[i,j,z] = (1/\rho_\ell) \cdot C^{(k_0,\ell)}[i,j]$. Then $\mathbb{E}[f_\ell[i,j,z] \mid H^{(\ell)}] = |\{k \in [n_2] : A[i,k] - A[i,k_0] = B[k_0,j] - B[k,j]\}| = |\{k \in [n_2] : A[i,k] + B[k,j] = A[i,k_0] + B[k_0,j] = z\}| = \text{COUNT}[i,j,z]$ by Fredman's trick. And $\text{Var}[f_\ell[i,j,z] \mid H^{(\ell)}] = (1/\rho_\ell^2) \cdot \text{COUNT}[i,j,z] \cdot (\rho_\ell - \rho_\ell^2) \leq 2^\ell t \cdot \text{COUNT}[i,j,z]$.

  Note that $f_\ell[i,j,z]$ (conditioned on a fixed $H^{(\ell)}$) depends on $R^{(\ell,k_0,\xi)}$ for $\xi = B[k_0,j] \bmod \Delta = z \bmod \Delta$. Thus, if $z \not\equiv z' \pmod{\Delta}$, then $f_\ell[i,j,z]$ and $f_\ell[i',j',z']$ are independent conditioned on any fixed $H^{(\ell)}$.

Finally, define $f[i,j,z]$ to be $f_\ell[i,j,z]$ for the smallest index $\ell \in [\log s]$ such that $W[i,j,z]$ is hit by $H^{(\ell)}$, or $f_*[i,j,z]$ if ℓ does not exist. Then $\mathbb{E}[f[i,j,z] \mid \{H^{(\ell)}\}_\ell] = \text{COUNT}[i,j,z]$ for any fixed $\{H^{(\ell)}\}_\ell$, which implies $\mathbb{E}[f[i,j,z]] = \text{COUNT}[i,j,z]$.

If $\text{COUNT}[i,j,z] \in [n_2/2^{p+1}, n_2/2^p)$ for $p \in [\log s]$, then $\ell \leq p+1$ w.h.p. Also, $\text{Var}[f[i,j,z] \mid \ell \leq p+1] \leq 2^{p+1} t \cdot \text{COUNT}[i,j,z]$ and $\text{Var}[f[i,j,z]$ is polynomially bounded regardless the value of ℓ, we have that $\text{Var}[f[i,j,z]] \leq O(2^{p+1} t \cdot \text{COUNT}[i,j,z]) \leq O(tn_2)$. Similarly, if $\text{COUNT}[i,j,z] < n_2/s$, then $\text{Var}[f[i,j,z]] \leq st \cdot \text{COUNT}[i,j,z] \leq O(tn_2)$. Note that for $(i,j) \neq (i',j')$ and $z \not\equiv z' \pmod{\Delta}$, $f[i,j,z]$ and $f[i',j',z']$ are independent conditioned on a fixed choice of $\{H^{(\ell)}\}_\ell$. Therefore, $\mathbb{E}[f[i,j,z]f[i',j',z'] \mid \{H^{(\ell)}\}_\ell] = \mathbb{E}[f[i,j,z] \mid \{H^{(\ell)}\}_\ell] \cdot \mathbb{E}[f[i',j',z'] \mid \{H^{(\ell)}\}_\ell] = \text{COUNT}[i,j,z] \cdot \text{COUNT}[i',j',z']$. Summing over all possible $\{H^{(\ell)}\}_\ell$ gives $\mathbb{E}[f[i,j,z]f[i',j',z']] = \text{COUNT}[i,j,z] \cdot \text{COUNT}[i',j',z'] = \mathbb{E}[f[i,j,z]] \cdot \mathbb{E}[f[i',j',z']]$, so $f[i,j,z]$ and $f[i',j',z']$ are uncorrelated.

The total time is

$$\begin{aligned}
&\widetilde{O}\left(\frac{n_1 n_2 n_3}{st} + \sum_{\ell=0}^{\log s} 2^\ell \Delta M_{\text{eq}}(n_1, n_2/(2^\ell t), n_3/\Delta)\right) \\
&\leq \widetilde{O}\left(\frac{n_1 n_2 n_3}{st} + \sum_{\ell=0}^{\log s} 2^\ell \Delta \left(\frac{n_1 n_2 n_3}{2^\ell ts\Delta} + M(n_1, sn_2/(2^\ell t), n_3/\Delta)\right)\right) \text{ by Lemma 2.3 with } r := s \\
&\leq \widetilde{O}\left(\frac{n_1 n_2 n_3}{st} + \sum_{\ell=0}^{\log s} 2^\ell \Delta \cdot (s/2^\ell) \cdot M(n_1, n_2/t, n_3/\Delta)\right).
\end{aligned}$$

□

3.2 Approximate Counting All-Numbers Convolution-3SUM

Next, we apply Lemma 3.1 to Convolution-3SUM, by modifying the known reduction [VW09] from Convolution-3SUM to Exact-Triangle. In the counting version of the All-Numbers Convolution-3SUM problem, we are

given 3 sequences $\langle a_0, \ldots, a_{n-1}\rangle$, $\langle b_0, \ldots, b_{n-1}\rangle$, and $\langle c_0, \ldots, c_{n-1}\rangle$, and want to count the number of k's such that $a_k + b_{h-k} = -c_h$, for each h. In the approximate counting version below, again the third sequence is unnecessary. Note that the time bound below is truly subquadratic ($\widetilde{O}(n^{(5+\omega)/4})$) for the case $t = \Delta = 1$.

Lemma 3.2. *Given integer sequences $\langle a_0, \ldots, a_{n-1}\rangle$ and $\langle b_0, \ldots, b_{n-1}\rangle$ where all a_k's are divisible by positive integer $\Delta \le n$, define*

$$\text{COUNT}[h, z] := |\{k \in [h] : a_k + b_{h-k} = z\}|.$$

Given parameter $1 \le t \le n$, there is a randomized algorithm that computes estimates $f[h, z]$ for all $h \in [n]$ and z, such that the expectation of $f[h, z]$ is equal to $\text{COUNT}[h, z]$, and the variance of $f[h, z]$ is $O(tn)$. (Zero entries of f need not be output explicitly.) Furthermore, $f[h, z]$ and $f[h', z']$ are uncorrelated if $h \ne h'$ and $z \not\equiv z' \pmod{\Delta}$. The running time is

$$\widetilde{O}\left(n^{(5+\omega)/4}/t^{(1+\omega)/4} + n^{(5+\omega)/4}\Delta^{(3-\omega)/4}/t + n\right).$$

Proof. Let d be an integer parameter between 1 and n. For simplicity, we assume n/d to be an integer, which can be achieved by padding ∞ entries to the arrays. We will estimate $\text{COUNT}^{(\ell)}[id + j, z] := |\{k \in [d] : a_{(i-\ell)d+k} + b_{\ell d+j-k} = z\}|$ for each $i \in [n/d]$, $j \in [d]$, and $\ell \in [n/d]$. (Assume that out-of-range array entries are set to ∞.)

Define an $(n/d) \times d$ matrix A and a $d \times d$ matrix $B^{(\ell)}$ for each $\ell \in [n/d]$: for each $i \in [n/d]$ and $k, j \in [d]$, let $A[i, k] = a_{id+k}$ and $B^{(\ell)}[k, j] = b_{\ell d+j-k}$. (Normally, it would be better to combine into one $d \times n$ matrix B and use rectangular matrix multiplication, but we need multiple independent subproblems here.) Apply Lemma 3.1 to estimate $\text{COUNT}^{(\ell)}[id + j, z] = |\{k \in [d] : A[i - \ell, k] + B^{(\ell)}[k, j] = z\}|$, for all $\ell \in [n/d]$, in total time

$$\begin{aligned}
&\widetilde{O}\left(\frac{n}{d} \cdot \left(\frac{(n/d) \cdot d \cdot d}{st} + s\Delta M(n/d, d/t, d/\Delta)\right)\right) \\
&= \begin{cases} \widetilde{O}\left(\frac{n^2}{st} + s\Delta\sqrt{n/t}M(\sqrt{n/t}, \sqrt{n/t}, \sqrt{tn}/\Delta)\right) & \text{if } t \ge \Delta \quad \text{by setting } d := \sqrt{tn} \\ \widetilde{O}\left(\frac{n^2}{st} + s\Delta\sqrt{n/\Delta}M(\sqrt{n/\Delta}, \sqrt{\Delta n}/t, \sqrt{n/\Delta})\right) & \text{if } t < \Delta \quad \text{by setting } d := \sqrt{\Delta n} \end{cases} \\
&\le \begin{cases} \widetilde{O}\left(\frac{n^2}{st} + s\Delta\sqrt{n/t} \cdot (t/\Delta) \cdot (\sqrt{n/t})^{\omega}\right) = \widetilde{O}\left(\frac{n^2}{st} + \frac{sn^{(\omega+1)/2}}{t^{(\omega-1)/2}}\right) & \text{if } t \ge \Delta \\ \widetilde{O}\left(\frac{n^2}{st} + s\Delta\sqrt{n/\Delta} \cdot (\Delta/t) \cdot (\sqrt{n/\Delta})^{\omega}\right) = \widetilde{O}\left(\frac{n^2}{st} + \frac{sn^{(\omega+1)/2}\Delta^{(3-\omega)/2}}{t}\right) & \text{if } t < \Delta \end{cases} \\
&= \begin{cases} \widetilde{O}\left(n^{(5+\omega)/4}/t^{(1+\omega)/4}\right) & \text{if } t \ge \Delta \quad \text{by setting } s := (n/t)^{(3-\omega)/4} \\ \widetilde{O}\left(n^{(5+\omega)/4}\Delta^{(3-\omega)/4}/t\right) & \text{if } t < \Delta \quad \text{by setting } s := (n/\Delta)^{(3-\omega)/4}. \end{cases}
\end{aligned}$$

We can now estimate $\text{COUNT}[h, z] = \sum_{\ell \in [n/d]} \text{COUNT}^{(\ell)}[h, z]$. The variance of each such estimate is $O((n/d) \cdot td) = O(tn)$, due to independence of the n/d applications of Lemma 3.1. ☐

3.3 Approximate Counting All-Numbers 3SUM

Next, we solve an analogous approximate counting version of 3SUM, by modify a known reduction from 3SUM to Convolution-3SUM by Chan and He [CH20b] (which preserves counts, unlike earlier reductions [Păt10, KPP16]). In the counting version of the All-Numbers 3SUM problem, we are given 3 sets A, B, and C, we want to count the number of $(a, b) \in A \times B$ such that $a + b = -c$, for each $c \in C$. Without the third set C, this amounts to computing counts/multiplicities of the elements of the sumset $A + B$.

Lemma 3.3. *Given sets A and B of n integers, where all elements of A are divisible by $\Delta \leq n$, define*

$$\text{COUNT}[z] := |\{(a,b) \in A \times B : a+b=z\}|.$$

Given parameter $1 \leq t \leq n$, there is a randomized algorithm that computes estimates $f[z]$ for all z, such that the expectation of $f[z]$ is equal to $\text{COUNT}[z]$, and the variance of $f[z]$ is $\widetilde{O}(tn)$. (Zero entries of f need not be output explicitly.) Furthermore, $f[z]$ and $f[z']$ are uncorrelated if $z \not\equiv z' \pmod{\Delta}$. The expected running time is

$$\widetilde{O}\left(n^{(5+\omega)/4}/t^{(1+\omega)/4} + n^{(5+\omega)/4}\Delta^{(3-\omega)/4}/t + n\right).$$

Proof. In one of the randomized reductions from 3SUM to Convolution-3SUM in [CH20b], one classifies all numbers as bad or good based on the choice of a hash function (it takes expected $O(n)$ time to do this preprocessing). Furthermore, the 3SUM instance between all the good elements can be reduced to $O(1)$ instances of Convolution-3SUM of size n, and if the size of the i-th set for $i \in [3]$ in the 3SUM instance is $\frac{n}{k_i}$, then the number of bad elements in it is at most $\frac{n}{2k_i^2}$. The same idea also works in our setting.

Say we have two sets of size $\frac{n}{k_1}$ and $\frac{n}{k_2}$ respectively, and say it takes $T(\frac{n}{k_1}, \frac{n}{k_2})$ time to solve such an instance. After we apply [CH20b]'s hash function in $O(n)$ expected time, the number of bad elements in the two sets becomes at most $\frac{n}{2k_1^2}$ and $\frac{n}{2k_2^2}$ respectively. For the good elements, we can apply Lemma 3.2. We then recursively solve the same problem between all bad elements in the first set, and all elements in the second set, which takes $T(\frac{n}{2k_1^2}, \frac{n}{k_2})$ time. Finally, we solve the same problem between all good elements in the first set and all bad elements in the second set, which takes $T(\frac{n}{k_1}, \frac{n}{2k_2^2})$. Clearly, summing up the results gives the correct expectation.

The running time can be written as the following recurrence:

$$T\left(\frac{n}{k_1}, \frac{n}{k_2}\right) \leq T\left(\frac{n}{2k_1^2}, \frac{n}{k_2}\right) + T\left(\frac{n}{k_1}, \frac{n}{2k_2^2}\right) + \widetilde{O}\left(n^{(5+\omega)/4}/t^{(1+\omega)/4} + n^{(5+\omega)/4}\Delta^{(3-\omega)/4}/t + n\right).$$

The recursion tree has depth $O(\log\log n)$, so it has size $2^{O(\log\log n)} = \log^{O(1)} n$. Therefore, the overall running time is

$$\widetilde{O}\left(n^{(5+\omega)/4}/t^{(1+\omega)/4} + n^{(5+\omega)/4}\Delta^{(3-\omega)/4}/t + n\right).$$

Say the variance bound for two sets of size $\frac{n}{k_1}$ and $\frac{n}{k_2}$ is $V\left(\frac{n}{k_1}, \frac{n}{k_2}\right)$, then we have the following recurrence (conditioned on any fixed hash function):

$$V\left(\frac{n}{k_1}, \frac{n}{k_2}\right) \leq V\left(\frac{n}{2k_1^2}, \frac{n}{k_2}\right) + V\left(\frac{n}{k_1}, \frac{n}{2k_2^2}\right) + O\left(tn\right),$$

which can be similarly upper bounded by $\widetilde{O}(tn)$. ∎

3.4 Approximate Counting Colored All-Numbers 3SUM

To solve the Text-to-Pattern Hamming Distances problem, we will actually need a colored version of counting 3SUM. This colored problem can be solved simply by independently invoking Lemma 3.3 for each color class. Note that the time bound below is sub-$n^{3/2}$ for the case $t = \Delta = 1$ and $U = n$.

Lemma 3.4. *Given sets A and B of n colored integers in $[U]$, where all elements of A are divisible by $\Delta \le n$, define*

$$\text{COUNT}[z] := |\{(a,b) \in A \times B : \ a+b=z, \ \text{color}(a) = \text{color}(b)\}|.$$

Given parameter $1 \le t \le n$, there is a randomized algorithm that computes estimates $f[z]$ for all z, such that the expectation of $f[z]$ is equal to $\text{COUNT}[z]$, and the variance of $f[z]$ is $\widetilde{O}(tn)$. Furthermore, $f[z]$ and $f[z']$ are uncorrelated if $z \not\equiv z' \pmod{\Delta}$. The expected running time is

$$\widetilde{O}\left(n(U/t)^{(1+\omega)/(5+\omega)} + nU^{(1+\omega)/(5+\omega)}\Delta^{(3-\omega)/(5+\omega)}/t^{4/(5+\omega)} + U\right).$$

Proof. Let A_c (resp. B_c) be the subset of all elements of A (resp. B) of color c. Let $n_c = |A_c| + |B_c|$. Estimate $\text{COUNT}_c[z] := |\{(a,b) \in A_c \times B_c : a+b=z\}|$ by Lemma 3.3 if $n_c \le x$, or compute it exactly by FFT in $\widetilde{O}(U)$ time if $n_c > x$. The number of calls to FFT is $O(n/x)$. The total time is

$$\begin{aligned}
&\widetilde{O}\left(\frac{nU}{x} + \sum_{n_c \le x}\left(n_c^{(5+\omega)/4}/t^{(1+\omega)/4} + n_c^{(5+\omega)/4}\Delta^{(3-\omega)/4}/t + n_c\right) + U\right) \\
&= \widetilde{O}\left(\frac{nU}{x} + nx^{(1+\omega)/4}/t^{(1+\omega)/4} + nx^{(1+\omega)/4}\Delta^{(3-\omega)/4}/t + U\right) \\
&= \widetilde{O}\left(n(U/t)^{(1+\omega)/(5+\omega)} + nU^{(1+\omega)/(5+\omega)}\Delta^{(3-\omega)/(5+\omega)}/t^{4/(5+\omega)} + U\right)
\end{aligned}$$

by setting $x := \min\{U^{4/(5+\omega)}t^{(1+\omega)/(5+\omega)}, \ (Ut)^{4/(5+\omega)}/\Delta^{(3-\omega)/(5+\omega)}\}$.

We can estimate $\text{COUNT}[z] = \sum_c \text{COUNT}_c[z]$, with variance $\widetilde{O}(\sum_c tn_c) = \widetilde{O}(tn)$ due to independence of the different applications of Lemma 3.3. ∎

The above lemma is sufficient to solve the approximate Text-to-Pattern Hamming Distances problem for distances that are $\Theta(m)$ (where we want additive error $O(\varepsilon m)$), but to deal with more generally distances that are $\Theta(k)$ (with additive error $O(\varepsilon k)$), we need to solve a generalization of the counting colored 3SUM problem where the input consists of $O(k)$ intervals (i.e., contiguous blocks of integers):

Lemma 3.5. *Given sets A and B of at most n colored integers in $[n]$, define*

$$\text{COUNT}[z] = |\{(a,b) \in A \times B : \ a+b=z, \ \text{color}(a) = \text{color}(b)\}|.$$

Suppose that A and B can be decomposed as unions of $O(k)$ disjoint intervals, where each interval is monochromatic. There is a randomized algorithm that computes estimates for $\text{COUNT}[z]$ for all z, with additive error $\widetilde{O}(\varepsilon k)$ w.h.p. The running time is

$$\widetilde{O}\left(\varepsilon^{-1+\beta}n^{1-\beta}k^{\beta} + \varepsilon^{-1-\beta}n^{1+\beta}/k^{2\beta} + n\right), \qquad \textit{where } \beta := (3-\omega)/(5+\omega).$$

Proof. We may assume that each interval has length at most n/k, by breaking the intervals; the number of intervals remains $O(k)$. Furthermore, we may assume that each interval is a *dyadic interval*, since each interval can be decomposed into a union of $O(\log n)$ dyadic intervals; the number of intervals remains $\widetilde{O}(k)$.

We may assume that each interval of A has the same length L and each interval of B has the same length ℓ, where L and ℓ are powers of 2 at most n/k, since we can try all pairs (L,ℓ) and solve $O(\log^2(n/k))$ instances; the time and additive error bounds increase only by polylogarithmic factors.

W.l.o.g., say $\ell \le L \le n/k$. Let A' (resp. B') denote the colored set of the $\widetilde{O}(k)$ left endpoints of the intervals in A (resp. B). Estimate $\textsc{count}'[z] = \{(a,b) \in A' \times B' : a+b=z,\ \mathrm{color}(a) = \mathrm{color}(b)\}$ by Lemma 3.4, with variance $\widetilde{O}(tk)$. Note that since endpoints in A' and B' are multiples of ℓ, the universe size can be shrunk to $U := n/\ell$, and after rescaling, all elements of A' are divisible by $\Delta := L/\ell \le n/(k\ell)$.

Let $w(i) := |\{(i_1,i_2) \in [L] \times [\ell] : i_1 + i_2 = i\}|$, i.e., $w(i) = \min\{i+1,\ \ell,\ L+\ell-i-1\}$ (for $i \in [L+\ell-1]$). Then $\textsc{count}[z] = \sum_{i=0}^{L+\ell-2} w(i)\textsc{count}'[z-i]$. From the estimates for $\textsc{count}'[\cdot]$, we can compute estimates for $\textsc{count}[\cdot]$ by doing an FFT in $\widetilde{O}(n)$ time (or by a more direct way, since $w(\cdot)$ is just a piecewise-linear function with 3 pieces). Note that $w(j) \le \ell$ and there are $O(L/\ell)$ nonzero terms $\textsc{count}'[z-i]$ in the sum; furthermore, after splitting the sum into two halves, in each half, no two $z-i$ values are equal mod L. Thus, we can estimate the sum of each half, with variance $O((L/\ell)\cdot\ell^2\cdot tk)$, due to pairwise uncorrelation. Then we can estimate $\textsc{count}[z]$ by summing up the two halves, which still has variance $O((L/\ell)\cdot\ell^2\cdot tk)$, as $\mathrm{Var}[X+Y] \le 2(\mathrm{Var}[X]+\mathrm{Var}[Y])$ for any random variables X and Y. This variance bound is $O((\varepsilon k)^2)$ by setting $t := \varepsilon^2 k/(L\ell) \ge \varepsilon^2 k^2/(n\ell)$. The running time is

$$\begin{aligned}
&\widetilde{O}\left(k\left(\frac{n/\ell}{\varepsilon^2k^2/(n\ell)}\right)^{(1+\omega)/(5+\omega)} + \frac{k(n/\ell)^{(1+\omega)/(5+\omega)}(n/(k\ell))^{(3-\omega)/(5+\omega)}}{(\varepsilon^2k^2/(n\ell))^{4/(5+\omega)}} + n\right)\\
&\le\ \widetilde{O}\left(\varepsilon^{-2(1+\omega)/(5+\omega)}n^{2(1+\omega)/(5+\omega)}k^{(3-\omega)/(5+\omega)} + \varepsilon^{-8/(5+\omega)}n^{8/(5+\omega)}/k^{2(3-\omega)/(5+\omega)} + n\right).
\end{aligned}$$

Since the estimate for $\textsc{count}[z]$ has variance $\widetilde{O}(tk) = \widetilde{O}((\varepsilon k)^2)$, the estimate has additive error $\widetilde{O}(\varepsilon k)$ with probability at least 0.9, say, by Chebyshev's inequality. We can repeat $\Theta(\log n)$ times and take the median, which achieves additive error $\widetilde{O}(\varepsilon k)$ w.h.p. by the Chernoff bound. □

3.5 Applying to Text-to-Pattern Hamming Distances

Finally, we connect the Text-to-Pattern Hamming Distances to colored counting 3SUM. The connection is not difficult to see: For each $a \in [n]$, put a in the set A with color $T[a]$, and for each $b \in [m]$, put b in the set B with color $P[b]$, where T and P are the given text and pattern strings. The number of matches between P and $T[i\,..\,i+m-1]$ is precisely $|\{(a,b) \in A \times B :\ a-b=i,\ \mathrm{color}(a) = \mathrm{color}(b)\}|$. So, we can apply Lemma 3.4 (with $U = n$, after negating B) to approximate the number of matches, and thus also the number of mismatches, with variance/additive error dependent on n.

However, we want additive error $O(\varepsilon k)$ for distances bounded by k. To this end, we apply a technique from known exact algorithms: Gawrychowski and Uznański [GU18] reduced k-bounded Text-to-Pattern Hamming Distances to $O(n/m)$ instance of the same problem for text and pattern strings of length $O(m)$ that have *run-length encodings* bounded by $O(k)$—in other words, the text and pattern strings are concatenations of $O(k)$ blocks of identical characters (*runs*). The reduction takes near-linear time, and preserves additive approximation. When mapped to the above colored sets A and B, these blocks become monochromatic intervals. So, we can apply Lemma 3.5 to approximate the number of matches with additive error $O(\varepsilon k)$, and thus the number of mismatches with additive error $O(\varepsilon k)$, and immediately obtain:

Theorem 3.6. *The approximate k-bounded Text-to-Pattern Hamming Distances problem additive error $O(\varepsilon k)$ can be solved by a Monte Carlo algorithm in time*

$$\widetilde{O}\left(\frac{n}{m}\cdot\left(\varepsilon^{-1+\beta}m^{1-\beta}k^{\beta} + \varepsilon^{-1-\beta}m^{1+\beta}/k^{2\beta} + m\right)\right), \qquad \textit{where } \beta := (3-\omega)/(5+\omega).$$

Theorem 3.6 implies Theorem 1.1, which we recall below:

Theorem 1.1 (Approximation algorithm with sublinear $1/\varepsilon$ dependence)**.** *The $(1+\varepsilon)$-approximate Text-to-Pattern Hamming Distances problem can be solved by a Monte Carlo randomized algorithm in $\widetilde{O}(\varepsilon^{-\gamma}n)$ time, where $\gamma = 8/(11-\omega) < 0.928$.*

Proof. For all shifts with Hamming distance $k \le \varepsilon^{-\gamma}\sqrt{m}$, we can use a known exact algorithm running in $\widetilde{O}(\frac{n}{m} \cdot (m + k\sqrt{m})) \le \widetilde{O}(\varepsilon^{-\gamma}n)$ time [GU18, CGK+20]. Otherwise, the bound in Theorem 3.6 is at most $\widetilde{O}(\frac{n}{m} \cdot (\varepsilon^{-1+\beta}m + \varepsilon^{-1-\beta+2\beta\gamma}m))$. We thus obtain an upper bound of $\widetilde{O}(\varepsilon^{-\gamma}n)$ in all cases, by setting $\gamma := (1+\beta)/(1+2\beta) = 8/(11-\omega) < 0.928$. We run the entire algorithm for every k that is a power of 2. □

Remark 3.7. With more tedious calculations, the exponent 0.928 can likely be improved by using known bounds on rectangular matrix multiplication, but the improvement would be tiny. If $\omega = 2$, the above bound is $\widetilde{O}(\varepsilon^{-8/9}n)$. Note that in the critical case when $k = \Theta(m)$ and $\varepsilon^{-1} = \sqrt{m}$ (which we have mentioned earlier), the bound in Theorem 3.6 is actually a little better: $\widetilde{O}(nm^{(1-\beta)/2})$, which is $\widetilde{O}(nm^{3/7})$ if $\omega = 2$.

4 Randomized Exact Text-to-Pattern Hamming Distances

In this and the next section, we turn to exact algorithms for Text-to-Pattern Hamming Distances.

4.1 $X + Y$ Lemma

The key ingredient of our new algorithm is the following lemma on computing the sumset $X + Y$, along with the multiplicities of its elements, for sets X and Y of n elements in a bounded integer universe (this is equivalent to computing convolutions of sparse binary vectors).

Lemma 4.1. *Given two (multi)sets X and Y of n elements in $[U/2]$ with $2U < n^2$, we can compute $\text{COUNT}[z] := |\{(x,y) \in X \times Y : x + y = z\}|$ for every $z \in [U]$ by a Las Vegas algorithm in $O(U \log(n^2/U))$ expected time.*

Note that Lemma 4.1 improves over the standard $O(U \log U)$ bound by FFT, when n^2 is not too large compared to U. Observe that the average count over all $z \in [U]$ is at most n^2/U, which is small in the regime of interest here. It is tempting to simply apply a bit-packed version of FFT (Lemma 2.2), but the challenge here is that there can be a mixture of elements with small and (possibly very) large counts in the sumset, and we don't know which elements have small or large counts in advance.

Our algorithm needs the almost-linear hash family by Dietzfelbinger [Die96], which was also used by Baran, Demaine, and Pătraşcu [BDP08] in their 3SUM algorithm. The following statement was proved in [BDP08].

Lemma 4.2 (Almost-linear hash family [BDP08])**.** *Let $L \le U$ be powers of two. There is a family of hash functions $f : [U] \to [L]$ with the following properties:*

(i) For all $x, y \in [U]$, $f(x) + f(y) - f(x+y) \in \Delta_f$ for some fixed set Δ_f of $O(1)$ size.

(ii) For any fixed x, y, z with $x + y \neq z$, $\mathbf{Pr}_f[f(x) + f(y) - f(z) \in \Delta_f] \le O(1/L)$.

(iii) Sampling and evaluating f take $O(1)$ time.

Although the above hash family has been used successfully for various problems related to convolutions, new technical issues arise when applying them to *counting* problems: even though we know that $h(x) + h(y) - h(x+y)$ lies in a set of $O(1)$ size for a hash function h, the precise value is still unpredictable. (If one uses another popular almost linear hash family, $h(x) = x \bmod p$ for random primes p, then this issue goes away, but collision probabilities increase by a $\log U$ factor, which we cannot afford to lose here.) Below, we prove new additional properties about (one version of) Dietzfelbinger's hash family, stating that for most "good" pairs (x, y), the value of $h(x) + h(y) - h(x + y)$ can be determined precisely by looking at some short labels $\tau_h(x)$ and $\tau_h(y)$ of x and y.

Lemma 4.3 (Almost-linear hash family with somewhat predictable errors)**.** *Let $q \le V \le U$ be powers of two. There is a family of hash functions $h\colon [U] \to [V]$ with the following properties:*

(i) For any fixed z, z' with $z \neq z'$, $\mathbf{Pr}_h[h(z) = h(z')] \le O(1/V)$.

(ii) There is a fixed set Δ_h of $O(1)$ size, and mappings $\tau_h\colon [U] \to [q^{O(1)}]$ and $\phi_h\colon [q^{O(1)}] \times [q^{O(1)}] \to \Delta_h \cup \{\textit{undefined}\}$ such that for all x, y, $h(x)+h(y)-h(x+y) = \phi_h(\tau_h(x), \tau_h(y))$ if $\phi_h(\tau_h(x), \tau_h(y))$ is defined. Call (x, y) good *if $\phi_h(\tau_h(x), \tau_h(y))$ is defined, and* bad *otherwise.*

(iii) For any fixed $x, y \in [U/2]$, $\mathbf{Pr}_h[(x, y)$ is bad$] \le O(1/q)$.

(iv) Sampling and evaluating h take $O(1)$ time. The mappings τ_h, ϕ_h can also be computed in $O(1)$ time.

Proof. For a nonnegative integer a, let $\mathrm{bin}(a)_i$ denote the i-th binary bit of a (for example, $\mathrm{bin}(a)_0 = a \bmod 2$). For $i \ge j \ge 0$, let $\mathrm{bin}(a)_{i:j}$ denote the concatenation $\mathrm{bin}(a)_i \circ \mathrm{bin}(a)_{i-1} \circ \cdots \circ \mathrm{bin}(a)_j$. Let $\mathrm{low}(a)$ denote the smallest i such that $\mathrm{bin}(a)_i = 1$; define $\mathrm{low}(0) = +\infty$.

Let $U = 2^w$, $V = 2^\ell$, and $q = 2^k$. We define the hash family as follows, similar to [Die96, BDP08]: Pick a random odd $r \in [2^{w+\ell}]$. For $x \in [2^w]$, define $h(x)$ as the following ℓ-bit integer,

$$h(x) = \mathrm{bin}(r \cdot x)_{w+\ell-1:w}.$$

Observe that, when $\mathrm{low}(x) = j$, we have $\mathrm{bin}(r \cdot x)_j = 1$, $\mathrm{bin}(r \cdot x)_{j'} = 0$ for all $j' < j$, and $\mathrm{bin}(r \cdot x)_{w+\ell-1:j+1}$ is a uniform random bit string.

We first prove Item (i). Given $0 \le z' < z < 2^w$, note that $h(z) = h(z')$ implies $(r \cdot z - r \cdot z') \equiv v \pmod{2^{w+\ell}}$ for some integer v with $|v| < 2^w$, which then implies $\mathrm{bin}(r \cdot (z - z'))_{w+\ell-1:w}$ is either the all-0 or all-1 ℓ-bit string. Since $\mathrm{low}(z - z') \le w - 1$, we know $\mathrm{bin}(r \cdot (z - z'))_{w+\ell-1:w}$ is a uniform random ℓ-bit string. Thus, $h(z) = h(z')$ happens with probability at most $2/2^\ell = O(1/V)$.

Now we prove Item (ii). First note that the error $h(x + y) - h(x) - h(y) \in \{0, 1, -2^\ell, -2^\ell + 1\}$, where the $+1$ term appears if and only if adding $r \cdot x$ and $r \cdot y$ generates a carry to the w-th bit, and the -2^ℓ term appears if and only if this addition generates a carry to the $(w + \ell)$-th bit. In order to predict these two carry bits, we can define τ_h as the following pair of k-bit strings,

$$\tau_h(x) = (\mathrm{bin}(r \cdot x)_{w-1:w-k},\ \mathrm{bin}(r \cdot x)_{w+\ell-1:w+\ell-k}).$$

Observe that adding $r \cdot x$ and $r \cdot y$ generates a carry to the w-th bit if and only if

$$\mathrm{bin}(r \cdot x)_{w-1:0} + \mathrm{bin}(r \cdot y)_{w-1:0} \ge 2^w.$$

By looking at the sums of their prefixes $\mathrm{bin}(r \cdot x)_{w-1:w-k}$ and $\mathrm{bin}(r \cdot y)_{w-1:w-k}$ (which are the first component in $\tau_h(x)$ and $\tau_h(y)$), we can unambiguously tell whether this inequality holds, except in the case

where this sum $\mathrm{bin}(r \cdot x)_{w-1:w-k} + \mathrm{bin}(r \cdot y)_{w-1:w-k}$ happens to equal the all-1 k-bit string. In this case we let $\phi_h(\tau_h(x), \tau_h(y))$ be undefined. Similarly, we use the second component of $\tau_h(x)$ and $\tau_h(y)$ to predict the carry to the $w+\ell$-th bit, and let $\phi_h(\tau_h(x), \tau_h(y))$ be undefined if this fails. This proves Item (ii).

Now we bound the probability that $\mathrm{bin}(r \cdot x)_{w-1:w-k} + \mathrm{bin}(r \cdot y)_{w-1:w-k}$ equals the all-1 k-bit string. Note that this event implies $\mathrm{bin}(r \cdot (x+y))_{w-1:w-k}$ is either the all-1 or all-0 string. We analyze each of these two cases, as follows:

- The all-1 case may happen only if $\mathrm{low}(r \cdot (x+y)) = \mathrm{low}(x+y) \le w-k$. In this case, $\mathrm{bin}(r \cdot (x+y))_{w-1:w-k+1}$ is a uniformly random $(k-1)$-bit string, so the probability that it equals the all-1 string is at most $1/2^{k-1} = O(1/q)$.
- Since we know $\mathrm{low}(r \cdot (x+y)) = \mathrm{low}(x+y) \le w-1$ from the assumption that $x, y \in [U/2]$, we know that the all-0 case may happen only if $\mathrm{low}(r \cdot (x+y)) = \mathrm{low}(x+y) \le w-k-1$. In this case, $\mathrm{bin}(r \cdot (x+y))_{w-1:w-k}$ is a uniformly random k-bit string, so the probability that it equals the all-0 string is at most $1/2^k = O(1/q)$.

Hence, we know the probability that $\mathrm{bin}(r \cdot x)_{w-1:w-k} + \mathrm{bin}(r \cdot y)_{w-1:w-k}$ equals the all-1 k-bit string is at most $O(1/q)$.

The probability that $\mathrm{bin}(r \cdot x)_{w+\ell-1:w+\ell-k} + \mathrm{bin}(r \cdot y)_{w+\ell-1:w+\ell-k}$ equals the all-1 string can be similarly bounded by $O(1/q)$. Then Item (iii) is proved by a union bound. □

Now we prove Lemma 4.1 by a new algorithm that uses a clever combination of hashing, bit-packed FFT, and recursion. We use hashing to reduce the number of live elements (elements whose counts are not yet known), but interestingly, we also use hashing and an extra recursive call to identify candidates for light elements (elements whose counts are small). Fortunately, since the number of live elements decreases at a super-exponential rate, the extra recursive calls do not blow up the final time bound, as we will see.

Proof. Our algorithm uses recursion. It recursively solves the following "partial version" of the original problem: There are M *live* elements $z \in [U]$ for which we are required to compute $\textsc{count}[z]$. For the remaining $U-M$ non-live elements $z \in [U]$, the correct values of $\textsc{count}[z]$ are already known and given to us. Let $T(n, M, U)$ be the expected time complexity of the problem of computing $\textsc{count}[z]$ for all the M live elements $z \in [U]$. The original problem corresponds to the case where all elements are live and $M = U$; so we want to bound $T(n, U, U)$.

Step 1: classify elements as light or heavy. Let L, s be parameters. Choose a random function $f : [U] \to [L]$ from a family of almost-linear hash functions from Lemma 4.2. Let $\textsc{count}_f[z] := |\{(x,y) \in X \times Y : f(x) + f(y) - f(z) \in \Delta_f\}|$. Call z *light* if $\textsc{count}_f[z] < 2sn^2/L$, and *heavy* otherwise. By Item (i) of Lemma 4.2, if z is light, then $\textsc{count}[z] \le \textsc{count}_f[z] < 2sn^2/L$.

To determine which elements are light or heavy, we recursively solve the problem for the (multi)sets $f(X)$ and $f(Y)$, in $T(n, O(L), O(L))$ time, and obtain $c_f[k] := |\{(x,y) \in X \times Y : f(x) + f(y) = k\}|$ for all k. Then we can obtain $\textsc{count}_f[z] = \sum_{\delta \in \Delta_f} c_f[f(z) + \delta]$.

We now bound the number of heavy live elements in two cases:

(i) First, since $\sum_z \textsc{count}[z] = |X||Y| = n^2$, the number of elements z with $\textsc{count}[z] \ge sn^2/L$ is at most L/s.

(ii) Now we fix a live element z with $\textsc{count}[z] < sn^2/L$. Note that $\textsc{count}_f[z] - \textsc{count}[z]$ is the number of $(x,y) \in X \times Y$ with $x + y \ne z$ and $f(x) + f(y) - f(z) \in \Delta_f$, which has expectation at

most $O(n^2/L)$ by Item (ii) of Lemma 4.2. By Markov's inequality, the probability that this number exceeds sn^2/L is $O(1/s)$. Thus, the probability that z is heavy is $O(1/s)$.

Summing up Case (i) and Case (ii) (over all M live elements), the expected number of heavy live elements is $O(L/s + M/s)$. By Markov's inequality, we can guarantee that this bound holds after $O(1)$ expected number of trials.

Step 2: classify elements as isolated or non-isolated. Let t and q be parameters. Independently choose another random function $h\colon [U] \to [tM]$ from a family of modified almost-linear hash functions from Lemma 4.3 (with parameter q). Let $\textsc{count}_{\text{live}}[z] := |\{z' \text{ live} : z' \neq z,\ h(z') = h(z)\}|$. Call a live element z *isolated* if $\textsc{count}_{\text{live}}[z] = 0$, and *non-isolated* otherwise.

For a fixed live element z, the expectation of $\textsc{count}_{\text{live}}[z]$ is $O(M \cdot 1/(tM)) = O(1/t)$ by Item (i) of Lemma 4.3. By Markov's inequality, the probability that z is non-isolated is $O(1/t)$. So, the expected number of non-isolated live elements is $O(M/t)$. By Markov's inequality, we can guarantee that this bound holds after $O(1)$ expected number of trials.

Step 3: compute counts for isolated light elements. Define

$$\begin{aligned} c[k] &:= \sum_{z:\ h(z)=k} \textsc{count}[z] \\ &= |\{(x,y) \in X \times Y : h(x+y) = k\}|. \end{aligned}$$

To compute $c[k]$, we decompose it into $c[k] = c_{\text{bad}}[k] + c_{\text{good}}[k]$ and compute separately, where $c_{\text{bad}}[k] := |\{(x,y) \in X \times Y \text{ bad} : h(x+y) = k\}|$ and $c_{\text{good}}[k] := |\{(x,y) \in X \times Y \text{ good} : h(x+y) = k\}|$ (see the definition of good and bad in Lemma 4.3). Let $X^{(\alpha)} := \{x \in X : \tau_h(x) = \alpha\}$ and $Y^{(\beta)} := \{y \in Y : \tau_h(y) = \beta\}$. We preprocess sets $X^{(\alpha)}$ for all $\alpha \in [q^{O(1)}]$ (and sets $Y^{(\beta)}$ for all $\beta \in [q^{O(1)}]$) in $O(n + q^{O(1)})$ time. Then,

- We can compute $c_{\text{bad}}[k]$ for all $k \in [tM]$ by examining each $\alpha, \beta \in [q^{O(1)}]$ such that $\phi_h(\alpha, \beta)$ is undefined, and enumerating all $(x,y) \in X^{(\alpha)} \times Y^{(\beta)}$, and incrementing the counter for $h(x+y)$. Since each pair $(x,y) \in X \times Y$ is bad with $O(1/q)$ probability by Item (iii) of Lemma 4.3, the expected total time is $O(q^{O(1)} + n^2/q)$.
- Fix a prime $p \in [2sn^2/L, 4sn^2/L]$.[8] We can compute $c_{\text{good}}[k] \bmod p$ for all $k \in [tM]$, by examining each $\alpha, \beta \in [q^{O(1)}]$ such that $\phi_h(\alpha, \beta)$ is defined, and computing $c_{\text{good}}^{(\alpha,\beta)}[k] := |\{(x,y) \in X^{(\alpha)} \times Y^{(\beta)} : h(x) + h(y) = k + \phi_h(\alpha,\beta)\}| \bmod p$, which reduces to a convolution problem for the multisets $h(X^{(\alpha)})$ and $h(Y^{(\beta)})$ in $[tM]$, done modulo p. By Lemma 2.2, each convolution takes $O(tM \log(sn^2/L))$ time. The total time over all α, β is $O(q^{O(1)} tM \log(sn^2/L))$.

Now, we know $c[k] \bmod p$ for all $k \in [tM]$ by summing up $c_{\text{bad}}[k]$ and $c_{\text{good}}[k] \bmod p$.

For each isolated live element z, we can compute $\textsc{count}[z] \bmod p$ by taking $c[h(z)]$ and subtracting $\textsc{count}[z']$ for all z' with $z' \neq z$ and $h(z') = h(z)$. Since z is isolated, all such elements z' are not live and thus their $\textsc{count}[z']$ values are known. The expected number of such elements z' is $O(U/(tM))$, by Item (i) of Lemma 4.3. Hence, the total expected time over all isolated live elements z is $O(M \cdot U/(tM)) = O(U/t)$.

If z is light, $\textsc{count}[z]$ is the same as $\textsc{count}[z] \bmod p$. Thus, we have computed $\textsc{count}[z]$ for all isolated light live elements z.

[8] The primes used by this recursive algorithm can be generated and fixed at the very beginning, in $\text{poly}\log(n)$ Las Vegas time.

Step 4: compute counts for non-isolated elements and heavy elements. The remaining live elements are non-isolated or heavy. We have already shown that there are $O(L/s + M/s + M/t)$ such elements. We recursively solve the problem for these elements in $T(n, O(L/s + M/s + M/t), U)$ time.

Analysis. We obtain the following recurrence:

$$\begin{aligned} T(n, M, U) &\leq O(1)T(n, O(L), O(L)) + T(n, O(L/s + M/s + M/t), U) \\ &\quad + O\big(q^{O(1)} tM \log(sn^2/L) + n^2/q + U/t\big). \end{aligned}$$

Let $M = U/r$. Set $L = U/r^{3/2}$, $s = t = \sqrt{r}$, and $q = r^{\varepsilon}$ for a sufficiently small constant $\varepsilon > 0$. Recall $n^2 > 2U$. Then the recurrence simplifies to

$$T(n, U/r, U) \leq O(1)T(n, O(U/r^{3/2}), U) + O\big((U/r^{1/2-O(\varepsilon)}) \log(n^2/U) + n^2/r^{\varepsilon}\big),$$

where we absorbed the r dependence in $\log(sn^2/L) = \log(r^2 n^2/U)$ into the $r^{O(\varepsilon)}$ factor. Expanding the recurrence gives

$$\begin{aligned} T(n, U/r, U) &\leq O\left(\sum_{i=0}^{\infty} O(1)^i \left((U/r^{(3/2)^i (1/2-O(\varepsilon))}) \log(n^2/U) + n^2/r^{(3/2)^i \varepsilon}\right)\right) \\ &= O\big((U/r^{1/2-O(\varepsilon)}) \log(n^2/U) + n^2/r^{\varepsilon}\big). \end{aligned} \tag{1}$$

Now we explain an alternative, simpler algorithm, to be used in the first level of recursion: Instead of running Steps 2–3, we just directly compute the counts for the light live elements by packed FFT (Lemma 2.2) over the universe U modulo a $p \in [2sn^2/L, 4sn^2/L]$, and recurse on the remaining heavy elements. The recurrence is

$$T(n, M, U) \leq O(1)T(n, O(L), O(L)) + T(n, O(L/s + M/s), U) + O(U \log(sn^2/L)).$$

Set $M = U$, $L = U/r_1$, and $s = r_1$. Then

$$\begin{aligned} T(n, U, U) &\leq O(1)T(n, O(U/r_1), U) + O(U \log(r_1^2 n^2/U)) \\ &\leq O((U/r_1^{1/2-O(\varepsilon)}) \log(n^2/U) + n^2/r_1^{\varepsilon} + U \log(r_1^2 n^2/U)) \qquad \text{by (1).} \end{aligned}$$

Finally, setting $r_1 = (n^2/U)^{1/\varepsilon}$ yields $T(n, U, U) = O(U \log(n^2/U))$. □

4.2 Text-to-Pattern Hamming Distances

Our exact algorithm for the Text-to-Pattern Hamming Distances problem (Theorem 1.2) now easily follows from Lemma 4.1.

Theorem 1.2 (Exact algorithm without log factors)**.** *The Text-to-Pattern Hamming Distances problem can be solved by a Las Vegas algorithm which terminates in $O(n\sqrt{m})$ time with high probability.*

Proof. For each character $c \in \Sigma$, let $A_c = \{a \in [n] : T[a] = c\}$ and $B_c = \{b \in [m] : P[b] = c\}$, and $n_c = |A_c| + |B_c|$. Note that $\sum_c n_c = O(n)$. The number of matches between P and $T[i \,..\, i + m - 1]$ is precisely $\sum_c |\{(a, b) \in A_c \times B_c : a - b = i\}|$. So, the problem reduces to solving an instance of the problem from Lemma 4.1 (after negating B_c) with n_c elements and universe size n, for each character c.

If $n_c \le \sqrt{2n}$, we can solve the problem by brute-force in $O(n_c^2)$ time. It is straightforward to bound the total running time of this case by $O(n^{3/2})$.

For $n_c > \sqrt{2n}$, we apply Lemma 4.1. Consider any $\ell = 1, \ldots, \lceil \log(\sqrt{n/2}) \rceil$, and consider n_c that is between $2^{\ell-1}\sqrt{2n}$ and $2^{\ell}\sqrt{2n}$. The number of such c is $O(\frac{\sqrt{n}}{2^\ell})$. Moreover, each time we call Lemma 4.1, if its running time exceeds twice its expectation, we rerun Lemma 4.1. By a standard application of the Chernoff bound, the total number of reruns is $O(\max\{\frac{\sqrt{n}}{2^\ell}, \log n\})$ w.h.p. Therefore, w.h.p., the running time contributed by these n_c is

$$O\left(\max\left\{\frac{\sqrt{n}}{2^\ell}, \log n\right\} \cdot n \cdot \log((2^\ell\sqrt{2n})^2/n)\right) = O\left(\frac{\ell}{2^\ell} \cdot n^{3/2} + \ell n \log n\right).$$

Summing up over all $\ell = 1, \ldots, \lceil \log(\sqrt{n/2}) \rceil$ gives the $O(n^{3/2})$ running time.

Finally, by breaking the problem into $O(n/m)$ instances of size $O(m)$, the time bound becomes $O((n/m) \cdot m^{3/2})$. □

4.3 k-Mismatch

Our technique also improves the previous algorithm for the k-mismatch problem. In fact, we only need to replace the use of FFT in [CGK+20]'s $O\big(n + \min\big(\frac{nk}{\sqrt{m}}\sqrt{\log m}, \frac{nk^2}{m}\big)\big)$-time algorithm with our new Lemma 4.1.

Theorem 1.5 (k-mismatch algorithm without log factors)**.** *The k-bounded Text-to-Pattern Hamming Distances problem can be solved by a Monte Carlo algorithm in $O\big(n + \frac{nk}{\sqrt{m}}\big)$ expected time which outputs correct answers with high probability.*

Proof Sketch. The bottleneck of [CGK+20]'s algorithm lies in the following task: given $2t$ sparse sequences $f_1, \ldots, f_t, g_1, \ldots, g_t$ whose supports are all in $[n]$, and the total size of their supports is $O(k)$, compute (a sparse representation of) $f_i \star g_i$ for every i. Additionally, all nonzero entries of f_i and g_i are either 0 or 1. The running time for this task in [CGK+20, Lemma 7.8] is $O(k \min(k, \sqrt{n \log n}))$. It suffices to improve it to provide an $O(k \min(k, \sqrt{n}))$.

Let n_i denote the sum of the support size of f_i and g_i. Note that $\sum_i n_i = O(k)$. We can either compute $f_i \star g_i$ using brute-force or using Lemma 4.1. Therefore the running time can be written as

$$O\left(\sum_{i:\ n_i \le \sqrt{2n}} n_i^2 + \sum_{i:\ n_i > \sqrt{2n}} n \log(n_i^2/n)\right) = O(k \min(k, \sqrt{n})).$$

□

4.4 Text-to-Pattern Dominance Matching

In the Text-to-Pattern Dominance Matching problem, we want to compute $|\{i : P[i] \le T[i+k]\}|$ for all k. For convenience in the following we solve the variant $|\{i : P[i] < T[i+k]\}|$ (which is without loss of generality).

We prove the first part of Theorem 1.4.

Theorem 4.4. *The Text-to-Pattern Dominance Matching problem can be solved by a Las Vegas algorithm which terminates in $O(n\sqrt{m})$ time with high probability.*

Proof. Let us sort all the $n + m \leq 2n$ characters (we treat the same character on different locations as different) of the text and the pattern together. For every dyadic interval I on this sorted array (without loss of generality, we assume the length of this array is a power of 2), let L be the set of indices of the characters in the pattern in the left half of the dyadic interval, and let R be the set of indices of the characters in the text in the right half of the dyadic interval. It suffices to count the contribution of $(i, j) \in L \times R$, i.e., the number of $(i, j) \in L \times R$ with $P[i] < T[j]$ and $i + k = j$ for every k whose count is nonzero. Because we sorted the characters, we already have $P[i] \leq T[j]$ for every $(i, j) \in L \times R$. If $\{P[i] : i \in L\}$ and $\{T[j] : j \in R\}$ share some character c (there can be at most one), let $L_c := \{i \in L : P[i] = c\}$ and $R_c := \{j \in R : T[j] = c\}$. Now it suffices to count the contributions from $(L \setminus L_c) \times R$ and $L_c \times (R \setminus R_c)$, each of which can be handled with convolution. Let the dyadic interval I be of length $2^{\ell+1}$. If $2^\ell \leq \sqrt{2n}$, we use brute-force to compute the convolution; otherwise, we use Lemma 4.1. Summing over all dyadic intervals give the following running time:

$$O\left(\sum_{0\leq\ell\leq\log(\sqrt{2n})} \frac{n}{2^\ell} \cdot (2^\ell)^2 + \sum_{\log(\sqrt{2n})<\ell\leq\log n} \frac{n}{2^\ell} \cdot n \log((2^\ell)^2/n)\right) = O(n\sqrt{n}).$$

Breaking the problem into $O(n/m)$ instances of size $O(m)$ gives the $O(n\sqrt{m})$ running time. As in the proof of Theorem 1.3, this can be made to hold w.h.p. by applying the Chernoff bound. □

5 Deterministic Exact Text-to-Pattern Hamming Distances

For our deterministic exact algorithm, we switch to a simpler approach to hashing, namely, taking a number mod m_i for some choice of m_i (instead of using Dietzfelbinger's hash family). We use the following known lemma by Chan and Lewenstein [CL15]:

Lemma 5.1 ([CL15])**.** *Given a set $T \subseteq [U]$ of size n, there exists an $n \cdot 2^{O(\sqrt{\log n \log\log U})} \cdot \mathrm{poly}\log(t, U)$ time deterministic algorithm that constructs $r = 2^{O(\sqrt{\log n \log\log U})}$ integers $m_1, \ldots, m_r = n \cdot 2^{\Theta(\sqrt{\log n \log\log U})} \cdot \mathrm{poly}\log(t, U)$, where for every $x \in T$, there exists $i \in [r]$ such that no other $y \in T$ has $y \equiv x \pmod{m_i}$.*

Note that we slightly adapted the results in [CL15] in that the integers $m_1, \ldots, m_r$ now also have a lower bound. This is without loss of generality because if some integer is too small, we can multiply it with an appropriate factor.

One particular application of Lemma 5.1 in [CL15] is a $t \cdot 2^{O(\sqrt{\log n \log\log U})} \cdot \mathrm{poly}\log(t, U)$ time deterministic algorithm for the sparse nonnegative convolution problem, in which we are given two sparse nonnegative sequences A, B, and we need to compute (a sparse representation of) their convolution $A \star B$, with the additional assumption that a small size-t superset of the support of the output sequence is given. Bringmann and Nakos [BN21a] removed this assumption via recursion. We first closely follow the approaches in [CL15] and [BN21a] to solve a problem similar to sparse nonnegative convolution.

Lemma 5.2. *Given three integer sequences A, B, C of length U, with the promise that $(A \star B - C)[i] \geq 0$ for every i, we can compute $A \star B$ in*

$$O(U) + t \cdot 2^{O(\sqrt{\log t \log\log U})} \cdot \mathrm{poly}\log(t, U)$$

deterministic time, where $t = \max\{||A||_0, ||B||_0, ||A \star B - C||_0\}$.

Proof. Without loss of generality, assume U is a power of 2. If $U = 1$, then the problem can be trivially solved in $O(1)$ time. In the following, we assume $U \geq 2$.

Let $U' = U/2$. For every $i \in [U']$, let $A'[i] := A[i] + A[i + U']$. We similarly prepare B' and C'. Then we recursively compute $A' \star B'$.

Given $A' \star B'$, we can compute $A' \star B' - C'$ in $O(U)$ time. Let S be the support of $A' \star B' - C'$. Notice that $(A' \star B' - C')[i] = (A \star B - C)[i] + (A \star B - C)[i + U'] + (A \star B - C)[i + 2U'] + (A \star B - C)[i + 3U']$. Therefore, by setting $T := \bigcup_{s \in S} \{s, s + U', s + 2U', s + 3U'\}$, T is guaranteed to be a superset of the support of $A \star B - C$. Furthermore, $|T| = O(||A \star B - C||_0) = O(t)$.

Next, we apply Lemma 5.1 on T and U and find $r = 2^{O(\sqrt{\log t \log\log U})}$ integers $m_1, \ldots, m_r = t \cdot 2^{\Theta(\sqrt{\log t \log\log U})} \cdot \operatorname{poly}\log(t, U)$. For each $k \in [r]$, we prepare two arrays A_k and B_k, which are defined as $A_k[i] := \sum_{j \equiv i \bmod m_k} A[j]$ and $B_k[i] := \sum_{j \equiv i \bmod m_k} B[j]$. Then we compute the following array D_k via FFT in $\widetilde{O}(m_k)$ time:

$$D_k[i] := \sum_{j \equiv i \bmod m_k} (A \star B)[j] = (A_k \star B_k)[i] + (A_k \star B_k)[i + m_k].$$

Furthermore, for each $x \in T$, we find the integer m_k such that for any other $y \in T$, $y \not\equiv x \pmod{m_k}$. The above takes $t \cdot 2^{O(\sqrt{\log t \log\log U})} \cdot \operatorname{poly}\log(t, U)$ time.

Next, for each $x \in T$ and the corresponding m_k, we compute the following value

$$E[x] := D_k[x \bmod m_k] - \sum_{j \equiv x \pmod{m_k}} C[j],$$

which equals

$$\sum_{j \equiv x \pmod{m_k}} (A \star B)[j] - \sum_{j \equiv x \pmod{m_k}} C[j] = \sum_{j \equiv x \pmod{m_k}} (A \star B - C)[j].$$

Since there is no other $y \in T$ such that $y \equiv x \bmod m_k$, and T is a superset of the support of $A \star B - C$,

$$\sum_{j \equiv x \pmod{m_k}} (A \star B - C)[j] = (A \star B - C)[x].$$

For any $x \notin T$, we can simply set $E[x]$ to be 0. The time for computing $E[x]$ for each $x \in T$ is $O(\frac{U}{m_k}) = O(\frac{U}{t \cdot 2^{\Theta(\sqrt{\log t \log\log U})} \cdot \operatorname{poly}\log(t,U)}) = O(\frac{U}{t})$, so the overall running time for computing E is $O(\frac{U}{t} \cdot |T|) = O(U)$.

Overall, $E = A \star B - C$, and we can compute $A \star B$ by adding E and C in $O(U)$ time.

The recursion adds an $\log(U)$ factor to the $t \cdot 2^{O(\sqrt{\log t \log\log U})} \cdot \operatorname{poly}\log(t, U)$ part of the running time. It only adds a constant factor to the $O(U)$ part of the running time, as U is halved at each recursion level. □

Lemma 5.3. *Given two (multi)sets X and Y of n elements in $[U/2]$ with $2U < n^2$, we can compute* $\textsc{count}[z] := |\{(x, y) \in X \times Y : x + y = z\}|$ *for every $z \in [U]$ by a deterministic algorithm in $O(U \log(n^2/U) + U\sqrt{\log n \log\log U})$ time.*

Proof. First, if $\frac{n^2}{U} = n^{\Omega(1)}$, then directly applying FFT already achieves the claimed running time. Now we assume $\frac{n^2}{U} = n^{o(1)}$.

Let p be a prime whose range is to be determined later. First, we use Lemma 2.2 to compute $\textsc{count}[z] \bmod p$ for every $z \in [U]$ in $O(U \log p)$ time.

Then we apply Lemma 5.2, using X for A, Y for B and $\text{COUNT}[z] \bmod p$ for C. Clearly, $||A \star B - C||_0 \leq \frac{n^2}{p}$, so the overall running time for computing $A \star B$ (which gives $\text{COUNT}[z]$) is $O(U) + t \cdot 2^{O(\sqrt{\log t \log\log U})} \cdot \operatorname{poly}\log(t, U)$ for $t = \max\{n, \frac{n^2}{p}\}$. Also, as $2U < n^2$, we can simplify the running time to $O\left(U + t \cdot 2^{c\sqrt{\log t \log\log U}}\right)$ for some constant c.

Finally, picking p from $\Theta\left(\frac{n^2}{U} \cdot 2^{c\sqrt{\log n \log\log U}}\right)$ (it only takes $n^{o(1)}$ time to find such a prime as $\frac{n^2}{U} = n^{o(1)}$) gives the desired running time. □

Theorem 1.3 (Deterministic exact algorithm)**.** *The Text-to-Pattern Hamming Distances problem can be solved by a deterministic algorithm in $O(n\sqrt{m}(\log m \log\log m)^{1/4})$ time.*

Proof. Let n_c be the number of occurrences of character c in the text and pattern. Note that $\sum_c n_c = O(n)$. As we know, the problem reduces to solving an instance of the problem from Lemma 5.3 with n_c elements and universe size n, for each character c.

For character c with $n_c \leq n^{1/2}(\log n \log\log n)^{1/4}$, we use the brute-force $O(n_c^2)$ time algorithm. For character c with $n_c > n^{1/2}(\log n \log\log n)^{1/4}$, we use the algorithm from Lemma 5.3 which runs in $O(n \log(n_c^2/n) + n\sqrt{\log n_c \log\log n})$ time. The overall running time is $O(n^{3/2}(\log n \log\log n)^{1/4})$.

Finally, by breaking the problem into $O(n/m)$ instances of size $O(m)$, the time bound becomes $O((n/m) \cdot m^{3/2}(\log m \log\log m)^{1/4})$. □

The deterministic algorithm for Text-to-Pattern Dominance Matching also easily follows from Lemma 5.2. Its proof is identical to the proof of Theorem 4.4, except that we replace Lemma 4.1 with Lemma 5.2 and update the running time analysis properly.

Theorem 5.4. *The Text-to-Pattern Dominance Matching problem can be solved by a deterministic algorithm in $O(n\sqrt{m}(\log m \log\log m)^{1/4})$ time.*

Remark 5.5. It would be natural to build Lemma 5.2 on Bringmann, Fischer and Nakos's more efficient algorithm [BFN22] for sparse nonnegative convolution that runs in $\widetilde{O}(t)$ time, in order to further improve our deterministic algorithm for exact Text-to-Pattern Hamming Distances. The difficulty in this approach is that, to implement [BFN22]'s idea, we have to view C as a degree U polynomial and evaluate it on t carefully chosen points. It is unclear how to do this evaluation in $o(U \log U)$ time ($O(U \log U)$ is the naive bound for Lemma 5.3 via FFT).

6 Equivalence with a Variant of 3SUM

In this section, we prove Theorem 1.6, which we recall below:

Theorem 1.6 (Equivalence with a variant of 3SUM)**.** *If Problem 2 has a $f(N)$ time algorithm, then Text-to-Pattern Hamming Distances with $n = O(m)$ has an $\widetilde{O}(f(m))$ time algorithm, and vice versa.*

Proof. The forward direction is implied by the proof of [BN20, Theorem 2.17]. For completeness, we include this simple proof. Suppose we have an algorithm $\mathcal{A}$ for Problem 2 in $T(N)$ time and we are given a Text-to-Pattern Hamming Distances instance with text T and pattern P where $n = O(m)$. For every $i \in [m]$, we add a number $-2nP[i] - i$ to a set A. For every $i \in [n]$, we add a number $2nT[i] + i$ to B. Finally, let $C = [n]$. Then we run algorithm $\mathcal{A}$ on sets A, B, C. It suffices to show that for every $i \in C$, the number of $(a, b) \in A \times B$ with $a + b = i$ is exactly the number of j where $P[j] = T[i + j]$ (if this is the

case, then the Hamming distance between P and $T[i\,..\,i+m-1]$ is m minus the count of 3SUM solutions for $i \in C$). In order for some number $-2nP[j]-j \in A$ and some number $2nT[k]+k \in B$ to sum up to i, we must have $P[j]=T[k]$ and $-j+k=i$. Therefore, the number of such pairs is exactly the number of j where $P[j]=T[i+j]$.

We next show the previously unknown backward direction. Suppose we have a $T(n)$-time algorithm $\mathcal{B}$ for Text-to-Pattern Hamming Distances with $n=O(m)$ and we are given an instance of Problem 2. First, we can negate all numbers in A, so that the task becomes finding the number of $(a,b) \in A \times B$ where $-a+b=c$ for every $c \in [N]$. Then we partition the sets A, B in the following way: for any integer g, let $A_g := \{a \in A : gN \le a < (g+1)N\}$ and similarly let $B_g := \{b \in B : gN \le b < (g+1)N\}$ (we do not need to create A_g or B_g that is empty). In order for $-a+b \in [N]$, we only need to match numbers in A_g with numbers in B_{g-1}, B_g, B_{g+1}. In the following, we only consider matching numbers in A_g with numbers in B_g, and the other two cases can be handled similarly.

For every g where A'_g and B'_g are nonempty, we sample a uniformly random shift $s_g \in [N]$. Let $A'_g := \{a - gN + s_g : a \in A_g\}$ and let $B'_g := \{b - gN + s_g : b \in B_g\}$ (this random shifts idea appeared in [LPW20]). Now the problem becomes, for every $c \in [N]$, find the number of $g, a \in A'_g, b \in B'_g$ where $-a+b=c$. Note that all numbers in A'_g and B'_g are in $[2N]$. For each $i \in [2N]$, the expected number of times it appears in A'_g and B'_g over all g is at most $\frac{1}{N}\sum_i(|A_g|+|B_g|) = O(1)$, so by the Chernoff bound, the number of times it appears in A'_g and B'_g is $O(\log N)$ w.h.p. For every $(x,y) \in \{1,\ldots,O(\log N)\}^2$, we create a Text-to-Pattern Hamming Distances instance as follows. Let the pattern P_x be of length $2N$, initially consisting of unique characters at each position. Then for every i, if A'_g is the x-th set (among all A'_g's) that contains i, we set $P_x[i]$ to be g. Similarly, let the text T_y be of length $3N$, initially consisting of unique characters at each position. Then for every i, if B'_g is the y-th set (among all B'_g's) that contains i, we set $T_y[i]$ to be g. Then we call algorithm $\mathcal{B}$ on each of these $O(\log^2 N)$ instances. For each shift $i \in [N]$, $2N$ minus the Hamming distance equals the number of j where $P_x[j]=T_y[i+j]$, i.e., it is the number of $g, a \in A'_g, b \in B'_g$ where $-a+b=i$, A'_g is the x-th set (among all A'_g's) that contains a, B'_g is the y-th set (among all B'_g's) that contains b. Summing over all (x,y) gives exactly the quantity we seek for each $i \in [N]$. The overall running time of this algorithm is $O(T(N)\log^2 N)$. □

7 Open Problems

We conclude with a few open questions:

- For $(1+\varepsilon)$-approximating Text-to-Pattern Hamming distances, what is the best possible dependence on $1/\varepsilon$? Are there *deterministic* algorithms faster than Karloff's $\widetilde{O}(\varepsilon^{-2}n)$ algorithm [Kar93]?
- Is there a $o(n\sqrt{m})$-time randomized algorithm for exact Text-to-Pattern Hamming Distances in the word-RAM model? Is there an $O(n\sqrt{m})$-time deterministic algorithm?
- Do our algorithms generalize to Text-to-Pattern ℓ_p Distances?

References

[Abr87] Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067. 1, 3, 5

[AD11] Mikhail J. Atallah and Timothy W. Duket. Pattern matching in the Hamming distance with thresholds. *Inf. Process. Lett.*, 111(14):674–677, 2011. doi:10.1016/j.ipl.2011.04.004. 3

[AF91] Amihood Amir and Martin Farach. Efficient matching of nonrectangular shapes. *Ann. Math. Artif. Intell.*, 4:211–224, 1991. doi:10.1007/BF01531057. 3

[AGW13] Mikhail J. Atallah, Elena Grigorescu, and Yi Wu. A lower-variance randomized algorithm for approximate string matching. *Inf. Process. Lett.*, 113(18):690–692, 2013. doi:10.1016/j.ipl.2013.06.005. 1

[ALP04] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X. 2, 5

[BC22] Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In *Proc. 49th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229, pages 31:1–31:21, 2022. doi:10.4230/LIPIcs.ICALP.2022.31. 2

[BCKL23] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast Fourier transform (ECFFT) part I: Low-degree extension in time $O(n \log n)$ over all finite fields. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 700–737, 2023. doi:10.1137/1.9781611977554.ch30. 31

[BDP08] Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. doi:10.1007/s00453-007-9036-3. 14, 15

[BF08] Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theor. Comput. Sci.*, 409(3):486–496, 2008. doi:10.1016/j.tcs.2008.08.042. 1

[BFN21] Karl Bringmann, Nick Fischer, and Vasileios Nakos. Sparse nonnegative convolution is equivalent to dense nonnegative convolution. In *Proc. 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1711–1724, 2021. doi:10.1145/3406325.3451090. 31

[BFN22] Karl Bringmann, Nick Fischer, and Vasileios Nakos. Deterministic and Las Vegas algorithms for sparse nonnegative convolution. In *Proc. 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3069–3090, 2022. doi:10.1137/1.9781611977073.119. 6, 22

[BN20] Karl Bringmann and Vasileios Nakos. Top-k-convolution and the quest for near-linear output-sensitive subset sum. In *Proc. 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 982–995, 2020. doi:10.1145/3357713.3384308. 4, 22

[BN21a] Karl Bringmann and Vasileios Nakos. Fast n-fold boolean convolution via additive combinatorics. In *Proc. 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198, pages 41:1–41:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.41. 20

[BN21b] Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In *Proc. 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1797–1815, 2021. doi:10.1137/1.9781611976465.108. 2

[BT09] Philip Bille and Mikkel Thorup. Faster regular expression matching. In *Proc. 36th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 171–182, 2009. doi:10.1007/978-3-642-02927-1_16. 1

[CFP+16] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142. 2, 5

[CGK+20] Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *Proc. 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 643–656, 2020. doi:10.1145/3357713.3384266. 1, 2, 3, 5, 6, 14, 19

[CH02] Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM J. Comput.*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527. 2, 5, 6

[CH20a] Timothy M. Chan and Qizheng He. On the change-making problem. In *Proc. 3rd SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 38–42, 2020. doi:10.1137/1.9781611976014.7. 31

[CH20b] Timothy M. Chan and Qizheng He. Reducing 3SUM to convolution-3SUM. In *Proc. 3rd Symposium on Simplicity in Algorithms (SOSA)*, pages 1–7, 2020. doi:10.1137/1.9781611976014.1. 6, 10, 11

[Cha10] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X. 5

[Cha18] Timothy M. Chan. Approximation schemes for 0-1 knapsack. In *Proc. 1st Symposium on Simplicity in Algorithms (SOSA)*, volume 61, pages 5:1–5:12, 2018. doi:10.4230/OASIcs.SOSA.2018.5. 2

[Cha20] Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median,+)-convolution, and some geometric 3SUM-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:https://doi.org/10.1145/3363541. 5

[CL15] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2015. doi:10.1145/2746539.2746568. 4, 6, 20

[Cli09] Raphaël Clifford. Matrix multiplication and pattern matching under Hamming norm, 2009. URL: https://web.archive.org/web/20160818144748/http://www.cs.bris.ac.uk/Re 1, 2, 3

[CLMZ23] Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. A nearly quadratic-time FPTAS for knapsack. *CoRR*, abs/2308.07821, 2023. arXiv:2308.07821, doi:10.48550/arXiv.2308.07821. 2

[CLZ03] Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM J. Comput.*, 32(6):1654–1673, 2003. doi:10.1137/S0097539702402007. 1

[CS98] David E. Cardoze and Leonard J. Schulman. Pattern matching for spatial point sets. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 156–165, 1998. doi:10.1109/SFCS.1998.743439. 31

[CVX23] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman's trick meets dominance product: Fine-grained complexity of unweighted APSP, 3SUM counting, and more. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 419–432. ACM, 2023. doi:10.1145/3564246.3585237. 4, 5, 6, 7, 8, 30

[Die96] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1046, pages 569–580, 1996. doi:10.1007/3-540-60922-9_46. 6, 14, 15

[DJM23] Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In *Proc. 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2961–2979, 2023. doi:10.1137/1.9781611977554.ch113. 2

[DWZ22] Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. *CoRR*, abs/2210.10173, 2022. To appear in FOCS 2023. arXiv:2210.10173, doi:10.48550/arXiv.2210.10173. 3

[FG13] Kimmo Fredriksson and Szymon Grabowski. Exploiting word-level parallelism for fast convolutions and their applications in approximate string matching. *Eur. J. Comb.*, 34(1):38–51, 2013. doi:10.1016/j.ejc.2012.07.013. 1

[FP74] Michael J. Fischer and Michael S. Paterson. String matching and other products. In *Complexity of Computation, RM Karp (editor), SIAM-AMS Proceedings*, volume 7, pages 113–125, 1974. 1, 5

[Fre76] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976. doi:10.1137/0205006. 5, 8

[Für14] Martin Fürer. How fast can we multiply large integers on an actual computer? In *Proc. 11th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 8392, pages 660–670. Springer, 2014. doi:10.1007/978-3-642-54423-1_57. 32

[GG86] Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17(4):52–54, 1986. doi:10.1145/8307.8309. 2, 5

[GP18] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018. doi:10.1145/3185378. 5

[Gra16] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discret. Appl. Math.*, 212:96–103, 2016. doi:10.1016/j.dam.2015.10.040. 1

[GS17] Omer Gold and Micha Sharir. Dominance product and high-dimensional closest pair under L_∞. In *Proc. 28th International Symposium on Algorithms and Computation (ISAAC)*, volume 92, pages 39:1–39:12, 2017. doi:10.4230/LIPIcs.ISAAC.2017.39. 30

[GU18] Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *Proc. 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62. 2, 4, 5, 6, 13, 14, 30

[HvdHL17] David Harvey, Joris van der Hoeven, and Grégoire Lecerf. Faster polynomial multiplication over finite fields. *J. ACM*, 63(6):52:1–52:23, 2017. doi:10.1145/3005344. 31, 32, 33, 34, 36, 37

[Ind98] Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proc. 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 166–173, 1998. doi:10.1109/SFCS.1998.743440. 1, 5, 7, 31

[Jin19] Ce Jin. An improved FPTAS for 0-1 knapsack. In *Proc. 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 76:1–76:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.76. 2

[JKS08] T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of Hamming distance. *Theory Comput.*, 4(1):129–135, 2008. doi:10.4086/toc.2008.v004a006. 2

[Kar93] Howard J. Karloff. Fast algorithms for approximately counting mismatches. *Inf. Process. Lett.*, 48(2):53–60, 1993. doi:10.1016/0020-0190(93)90177-B. 1, 4, 5, 23

[KP15] Tsvi Kopelowitz and Ely Porat. Breaking the variance: Approximating the Hamming distance in $\tilde{O}(1/\varepsilon)$ time per alignment. In *Proc. IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 601–613, 2015. doi:10.1109/FOCS.2015.43. 2, 3, 5

[KP18] Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern Hamming distance. In *Proc. 1st Symposium on Simplicity in Algorithms (SOSA)*, volume 61, pages 10:1–10:5, 2018. doi:10.4230/OASIcs.SOSA.2018.10. 2, 3, 4, 5

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1287, 2016. doi:10.1137/1.9781611974331.ch89. 10

[LAHC16] Sian-Jheng Lin, Tareq Y. Al-Naffouri, Yunghsiang S. Han, and Wei-Ho Chung. Novel polynomial basis with fast fourier transform and its application to reed-solomon erasure codes. *IEEE Trans. Inf. Theory*, 62(11):6284–6299, 2016. doi:10.1109/TIT.2016.2608892. 7, 31

[LP11] Ohad Lipsky and Ely Porat. Approximate pattern matching with the L_1, L_2, and L_∞ metrics. *Algorithmica*, 60(2):335–348, 2011. doi:10.1007/s00453-009-9345-9. 2

[LPW20] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *Proc. 11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 53:1–53:18, 2020. doi:10.4230/LIPIcs.ITCS.2020.53. 4, 23, 30

[LU18] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046, 2018. doi:10.1137/1.9781611975031.67. 30

[LUW19] Karim Labib, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Hamming distance completeness. In *Proc. 30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 128, pages 14:1–14:17, 2019. doi:10.4230/LIPIcs.CPM.2019.14. 3, 4, 30

[LV86] Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7. 2, 5

[LV89] Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2. 2, 5

[Mao23] Xiao Mao. (1-ε)-approximation of knapsack in nearly quadratic time. *CoRR*, abs/2308.07004, 2023. arXiv:2308.07004, doi:10.48550/arXiv.2308.07004. 2

[Mat91] Jiří Matoušek. Computing dominances in E^n. *Inf. Process. Lett.*, 38(5):277–278, 1991. doi:10.1016/0020-0190(91)90071-O. 4, 7, 30

[MP80] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1. 1

[MWW19] Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for partition. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 70–88, 2019. doi:10.1137/1.9781611975482.5. 2

[Mye92] Gene Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 39(2):432–448, apr 1992. doi:10.1145/128749.128755. 1

[Păt10] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, pages 603–610, 2010. doi:10.1145/1806689.1806772. 10

[Sho88] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–290, 1988. doi:10.1109/SFCS.1988.21944. 33, 36

[SU19] Jan Studený and Przemysław Uznański. Approximating approximate pattern matching. In *Proc. 30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 128, pages 15:1–15:13, 2019. doi:10.4230/LIPIcs.CPM.2019.15. 2

[SV96] Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *Proc. 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 320–328, 1996. doi:10.1109/SFCS.1996.548491. 2, 5

[Tak98] Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998. doi:10.1007/PL00009198. 5

[Tho02] Mikkel Thorup. Randomized sorting in $O(n \log \log n)$ time and linear space using addition, shift, and bit-wise boolean operations. *J. Algorithms*, 42(2):205–230, 2002. doi:10.1006/jagm.2002.1211. 35

[Uzn20a] Przemysław Uznański. Approximating text-to-pattern distance via dimensionality reduction. In *Proc. 31st Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 161, pages 29:1–29:11, 2020. doi:10.4230/LIPIcs.CPM.2020.29. 2

[Uzn20b] Przemysław Uznański. Recent advances in text-to-pattern distance algorithms. In *Beyond the Horizon of Computability - 16th Conference on Computability in Europe (CiE)*, volume 12098, pages 353–365, 2020. doi:10.1007/978-3-030-51466-2_32. 2, 4

[Vas15] Virginia Vassilevska Williams. Problem 2 on problem set 2 of CS367, October 15, 2015. URL: http://theory.stanford.edu/~virgi/cs367/hw2.pdf. 4, 30

[VW09] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 455–464, 2009. doi:10.1137/09076619X. 4, 6, 9

[VW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893. 8

[VXXZ23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. *CoRR*, abs/2307.07970, 2023. To appear in SODA 2024. arXiv:2307.07970, doi:10.48550/arXiv.2307.07970. 3, 30

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2013. 37

[WC22] Xiaoyu Wu and Lin Chen. Improved approximation schemes for (un-)bounded subset-sum and partition. *CoRR*, abs/2212.02883, 2022. arXiv:2212.02883, doi:10.48550/arXiv.2212.02883. 2

[Wil18] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524. 5

[Woo04] David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175, 2004. URL: https://dl.acm.org/doi/10.5555/982792.982817. 2

[Yus09] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 950–957, 2009. URL: https://dl.acm.org/doi/10.5555/1496770.1496873. 7, 30

A Slight Extension to Indyk's Reduction

In this appendix, we observe a simple extension to Indyk's reduction from BMM to Text-to-Pattern Hamming Distances, so that instead of BMM we start from the following Equality Product problem: Given two $N \times N$ integer matrices A and B, compute the $N \times N$ matrix C where $C[i,j] = |\{k \mid A[i,k] = B[k,j]\}|$.

Equality Product has been studied in several papers [Mat91, LUW19, Vas15, CVX23, GS17, LPW20]. The fastest known algorithm [Yus09] for it runs in $O(N^{2.659})$ time using rectangular matrix multiplication [LU18, VXXZ23]. This running time would be $O(N^{2.5})$ if $\omega = 2$.

Equality Product is among the so called "intermediate" matrix products which seem to require $N^{2.5-o(1)}$ time (in the word-RAM model of computation with $O(\log N)$ bit words), even if $\omega = 2$ (see [LPW20, LUW19]).

Here we reduce Equality Product to Text-to-Pattern Hamming Distances, following Indyk's reduction closely.

Given $N \times N$ matrices A and B, we create a text T and a pattern P, both of length $\Theta(N^2)$ as follows.

First let us define our alphabet Σ. For every $i,j \in [N]$, interpret $(j, A[i,j])$ as a new letter in Σ. Similarly, for every $j,k \in [N]$, add letters $(j, B[j,k])$ to Σ. So far, Σ is a subset of $[n] \times \mathbb{Z}$. Also let \$ be a new letter that does not appear in Σ so far, adding it to Σ.

Encode each row A_i of A as a string $f(i) = (0, A[i,0]) \odot (1, A[i,1]) \odot \ldots \odot (N-1, A[i,N-1])$, where $\odot$ means concatenation. Let the text be

$$T = \$^{N^2} \odot f(0) \odot \$ \odot f(1) \odot \$ \odot \ldots \odot \$ \odot f(N-1) \odot \$^{N^2}.$$

Similarly, encode each column B_k of B as a string $g(k) = (0, B[0,k]) \odot (1, B[1,k]) \odot \ldots \odot (N-1, B[N-1,k])$. Let the pattern be

$$P = g(0) \odot g(1) \odot \ldots g(N-1).$$

Note that the Hamming distance between $f(i)$ and $g(k)$ is exactly the number of j for which $A[i,j] \neq B[j,k]$, so that $N-$the Hamming distance of $f(i)$ and $g(k)$ is exactly the number of j for which $A[i,j] = B[j,k]$.

Similarly to Indyk's reduction, the \$ symbols in T ensure that if we align P with T so that $f(i)$ is exactly aligned with $g(k)$, then there are no other symbols of Σ that can be equal and aligned except those in $f(i)$ and $g(k)$, and so the Hamming distance between T and P for the corresponding shift equals $|P|-$the number of j for which $A[i,j] = B[j,k]$.

The lengths n and m of T and P are both $\Theta(N^2)$. Thus, any algorithm that runs in $O(nm^{1/4-\varepsilon})$ time for $\varepsilon > 0$ for Text-to-Pattern Hamming Distances would result in an $O(N^2 \cdot N^{2/4-2\varepsilon}) = O(N^{2.5-2\varepsilon})$ time algorithm for Equality Product.

The lower bound for Text-to-Pattern Hamming Distances would be higher if we assumed that the current best known algorithms for Equality Product are optimal. In particular, if Equality Product requires $N^{2.5+\delta-o(1)}$ time for some $\delta > 0$, then the lower bound for Text-to-Pattern Hamming Distances becomes $nm^{1/4+\delta/2-o(1)}$.

Extension to Gawrychowski and Uznański's reduction for k-mismatch. We remark that the same modification can be performed on the reduction by Gawrychowski and Uznański [GU18] for the k-mismatch problem, to give a conditional lower bound for k-mismatch of $((kn/\sqrt{m}) \cdot (1/m^{1/4}))^{1-o(1)}$ which would hold even if $\omega = 2$. Recall that the fastest algorithm runs in $O(n + kn/\sqrt{m})$ time.

Gawrychowski and Uznański presented a reduction from Boolean Matrix Multiplication of an $M' \times N$ matrix by an $N \times M$ matrix (for $M' \geq M \geq N$), to the k-mismatch problem for a text of length n and a pattern of length m where $m = M^2$, $n = M'M$ and $k = MN$.

Using our simple modification we immediately obtain a reduction from the Equality Product of an $M' \times N$ matrix by an $N \times M$ matrix to the k-mismatch problem for a text of length n and a pattern of length m where $m = M^2$, $n = M'M$ and $k = MN$.

The fastest algorithm for this rectangular Equality Product when $M \leq N^2$ runs in $O(M'N\sqrt{M})$ time if $\omega = 2$.

Suppose that k-mismatch has an algorithm running in time $O((kn/m^{3/4})^{1-\varepsilon})$ time for some $\varepsilon > 0$. Then Equality Product of an $M' \times N$ matrix by an $N \times M$ matrix can be solved asymptotically in time

$$\left(\frac{(MN)\cdot(M'M)}{(M^2)^{3/4}}\right)^{1-\varepsilon} = \left(M'N\sqrt{M}\right)^{1-\varepsilon},$$

thus beating the best known running time for rectangular Equality Product in this setting, even if $\omega = 2$.

B Polynomial Multiplication over $\mathbb{F}_p$ in word RAM

In this section, we prove Lemma 2.2, which we recall below:

Lemma 2.2. *Given a prime $p \leq n^{O(1)}$ and two length-n sequences a, b with entries in $\mathbb{F}_p$, we can deterministically compute $a \star b$ in $O(n \log p)$ time.*

Indyk's original approach. Indyk [Ind98] claimed a proof of Lemma 2.2 (originally described in the $p = 2$ case) as follows: in the word RAM model with $\Theta(\log n)$-bit word length, we can pack $\ell = \Theta(\frac{\log n}{\log p})$ numbers in $\mathbb{F}_p$ to a single word, represented in $\mathbb{F}_{p^{2\ell-1}}$, which reduces the length of the arrays to $O(n/\ell)$. Then we essentially need to multiply two degree-$O(n/\ell)$ polynomials over $\mathbb{F}_{p^{2\ell-1}}$. Indyk [Ind98] assumed that this multiplication could be done in $O((n/\ell)\log(n/\ell))$ time (where each field operation in $\mathbb{F}_{p^{2\ell-1}}$ takes constant time), which would imply the desired $O(n \log p)$ time bound. Today, it is known how to perform this multiplication for $p = 2$ in $O((n/\ell)\log(n/\ell))$ time [LAHC16], but for general p, the current best algorithm for multiplying two degree-m polynomials over a finite field uses $O(m \log m \cdot 2^{O(\log^* m)})$ field operations [HvdHL17] (for finite fields with primitive roots of large smooth order, the textbook Cooley-Tukey FFT has a faster $O(m \log m)$ run time, but for general finite fields it is a major open question to remove this $2^{O(\log^* m)}$ factor; see e.g., discussion in [BCKL23]), so Indyk's original proof (combined with the state-of-the-art [HvdHL17] result directly as a black box) only implies an algorithm with slower $O(n \log p \cdot 2^{O(\log^* n)})$ time.

Remark B.1. We briefly discuss other papers that relied on Indyk's algorithm and are hence affected by this issue (but can be saved either by [LAHC16] or using our new proof of Lemma 2.2). [CS98] studied pattern matching for point sets and gave a randomized $O(n \log n)$ time algorithm. [CH20a] studied the coin change problem and gave a randomized $O(t \log t)$ time decision algorithm. [BFN21] gave a randomized $O(k \log k)$ time non-negative sparse convolution algorithm. These results [CS98, CH20a, BFN21] only use the case $p = 2$, so they can already be fixed by [LAHC16].

In the rest of the section, we describe a more involved word RAM algorithm that saves this additional $2^{O(\log^* n)}$ factor, proving Lemma 2.2. It builds on the recursive algorithm of [HvdHL17] and additionally uses bit tricks and table look-ups (in a similar spirit to the $O(n)$-time integer multiplication algorithm in the

word RAM model by [Für14]). This algorithm does not solve the aforementioned major question left open by [HvdHL17], since it runs on the word RAM model.

First, multiplying two length-n polynomials over $\mathbb{F}_p$ can be reduced to multiplying two $O(n \log(pn))$-bit integers via a standard Kronecker substitution (see [HvdHL17, Section 2.6]). The latter task can be done in $O(n \log(pn))$ time in word RAM [Für14] using FFT with bit packing. If $p \geq n^{0.01}$, then the running time is $O(n \log(pn)) = O(n \log p)$ as desired. Hence, in the rest of the section we assume $p \leq n^{0.01}$.

At a high level, our algorithm uses the techniques from [HvdHL17]'s $\mathbb{F}_p$-polynomial multiplication algorithm with $O(n \log n \cdot 8^{\log^* n} \cdot \log p)$ bit complexity in the Turing Machine model. Their algorithm has roughly $\log^* n$ levels of recursion, where each level exponentially decreases the length of the DFT. Here we adapt their algorithm to the word RAM model: we only need two levels of recursion to decrease the length of the DFT to sub-logarithmic, and then we look up the DFT results from preprocessed tables. We also need to use some bit tricks to speed up the DFT implementation.

Number-theoretic lemmas. We call a positive integer y-*smooth* if all of its prime divisors are less than or equal to y. We quote the following two theorems from [HvdHL17].

Lemma B.2 ([HvdHL17, Theorem 4.1])**.** *There exist computable absolute constants $c_3 > c_2 > 0$ and $n_0 \in \mathbb{N}$ with the following properties. Let p be a prime, and let $n \geq n_0$. Then there exists an integer λ in the interval*

$$(\log n)^{c_2 \log\log\log n} < \lambda < (\log n)^{c_3 \log\log\log n},$$

and a $(\lambda+1)$-smooth integer $M \geq n$, such that $M \mid p^\lambda - 1$. Moreover, given p and n, we may compute λ and the prime factorization of M in time $O((\log n)^{\log\log n})$.

Lemma B.3 ([HvdHL17, Theorem 4.6])**.** *Let p, n, λ, M be as in Lemma B.2. Let R and S be positive integers such that $\lambda < S < R < M$. Then there exist $(\lambda+1)$-smooth integers $m_1, \ldots, m_d$ with the following properties:*

1. *$N := m_1 \cdots m_d$ divides M (and hence divides $p^\lambda - 1$).*
2. *$R \leq N \leq (\lambda+1)R$.*
3. *$S \leq m_i \leq S^3$ for all i.*

Given λ, S, R, and the prime factorization of M, we may compute such $m_1, \ldots, m_d$ (and their factorizations) in time $\tilde{O}(\lambda^3)$.

We will use the following corollary which combines the two lemmas above.

Corollary B.4. *Let p be a prime, and let $n \geq n_0$ (where n_0 is an absolute constant). Then there exist integers $n' \in [n, 2n]$, $L \in (\log n)^{\Theta(\log\log\log n)}$, and $m_1, \ldots, m_d$ (all computable in $n^{o(1)}$ time), such that:*

- *$N := m_1 \cdots m_d$ divides $p^L - 1$,*
- *$n' = NL$, and*
- *$\sqrt{L}/2 \leq m_i \leq L^3$.*

Proof. Apply Lemma B.2 with p, n and obtain λ, M. Then apply Lemma B.3 with $S = \lambda + 1$ and $R = \lfloor n/(\lambda+1)^2 \rfloor$ to obtain $N := m_1 \cdots m_d$.

Let n' be the smallest $n' \geq n$ such that n' is a multiple of $N\lambda$. Since $N\lambda \leq (\lambda+1)R \cdot \lambda \leq (n/(\lambda+1)) \cdot \lambda < n$, we have $n' \leq 2n$. Define $L := n'/N$. Since $N\lambda \mid n'$, we must have $\lambda \mid L$, and hence $N \mid p^\lambda - 1 \mid p^L - 1$.

From $R \leq N \leq (\lambda+1)R$ (where $R = \lfloor n/(\lambda+1)^2 \rfloor$) we have $n/N \in [\lambda+1, 2(\lambda+1)^2]$, and hence $L = n'/N \in [n/N, 2n/N] \subseteq [\lambda+1, 4(\lambda+1)^2] \subseteq (\log n)^{\Theta(\log\log\log n)}$.

From $S \leq m_i \leq S^3$, $S = \lambda + 1$, and $L \in [\lambda+1, 4(\lambda+1)^2]$, we have $\sqrt{L}/2 \leq m_i \leq L^3$.

The running time for applying the two lemmas is $O((\log n)^{\log\log n}) + \tilde{O}(\lambda^3) \leq O((\log n)^{\log\log n}) \leq n^{o(1)}$. □

Now we describe the parameters in the two levels of our algorithm for multiplying two degree-n polynomials over $\mathbb{F}_p$. We will invoke Corollary B.4 twice with two different n's (and always the same p).

First level parameters. Let $n' = N_1 L_1$ be returned by Corollary B.4 when plugging in n. Here $n' \in [n, 2n]$, $L_1 \in (\log n)^{\Theta(\log\log\log n)}$, and $N_1 \mid p^{L_1} - 1$.

We will consider the sub-problem of multiplying two degree-L_1 polynomials over $\mathbb{F}_p$. At this point, we first address the easy case where $p \geq L_1^{\Omega(1)}$, which can be solved without the second-level recursion: we simply do a Kronecker substitution to reduce this sub-problem to polynomial multiplication with integer coefficients.

Lemma B.5 (Second level (degenerate case))**.** *Suppose $p > L_1$. Then multiplying two degree-L_1 polynomials over $\mathbb{F}_p$ can be done in $O((L_1 \log L_1) \cdot \frac{\log p}{\log n})$ time.*

Proof. In this case, we may use standard Kronecker substitution to pack the coefficients into large integers (see [HvdHL17, Section 2.6]). More specifically, we can pack b coefficients in $\mathbb{F}_p$ into an integer of magnitude $O(p^2 L_1)^b = p^{O(b)}$ as $p > L_1$. To fit each integer in a word, we can set $p^{O(b)} = n^{O(1)}$, so b can be as large as $O(\frac{\log n}{\log p})$. Then the problem reduces to multiplying two degree-$\frac{L_1}{b}$ polynomials with integer coefficients (each fitting into one word), which can be done using FFT in $O(\frac{L_1}{b}\log(\frac{L_1}{b})) = O((L_1 \log L_1) \cdot \frac{\log p}{\log n})$ time as desired. □

In the following, we need to prove Lemma B.5 in the hard case where $p \leq L_1$, via a second level of recursion.

Second level parameters. Assume $p \leq L_1^{O(1)}$. Let $L_1' = N_2 L_2$ be returned by Corollary B.4 when plugging in L_1 in place of n. Here $L_1' \in [L_1, 2L_1]$ and $L_2 \in (\log L_1)^{\Theta(\log\log\log L_1)} \subseteq (\log\log n)^{\Theta(\log^{(4)} n)}$.

In the following we will work over the finite field $\mathbb{F}_{p^{L_2}}$. Note that we can find a representation of $\mathbb{F}_{p^{L_2}}$ by finding an irreducible monic polynomial of degree L_2, which can be done in $\widetilde{O}(L_2^4 p^{1/2}) \leq \widetilde{O}(L_2^4 L_1^{1/2}) = n^{o(1)}$ time deterministically [Sho88]. Since $N_2 \mid p^{L_2} - 1$ by Corollary B.4, we can find a primitive N_2-th root of unity ω_{N_2} in $\mathbb{F}_{p^{L_2}}$ in time $\widetilde{O}(L_2^9 p) = n^{o(1)}$ [HvdHL17, Lemma 3.3]. Note that for any factor m of N_2, $\omega_m := \omega_{N_2}^{N_2/m} \in \mathbb{F}_{p^{L_2}}$ is a primitive m-th root of unity, and recall the DFT of an length-m array $(a_0, \ldots, a_{m-1}) \in (\mathbb{F}_{p^{L_2}})^m$ is the array $(\hat{a}_0, \ldots, \hat{a}_{m-1}) \in (\mathbb{F}_{p^{L_2}})^m$ where $\hat{a}_k := \sum_{j=0}^{m-1} a_j \cdot \omega_m^{jk}$.

Let $N_2 = m_1' \cdots m_{d'}'$ as in Corollary B.4, where $m_i' \in L_2^{\Theta(1)} \subseteq (\log\log n)^{\Theta(\log^{(4)} n)}$. Since we assumed $p \leq L_1^{O(1)}$, we have $m_i' L_2 \log p \leq L_2^{O(1)} L_2 \log L_1 \leq (\log\log n)^{O(\log^{(4)}(n))} < 0.1 \log n$. We also know

$N_2 = \Theta(L_1/L_2) \gg \log n$. Hence, by greedily grouping the factors m_i', we can get a factorization

$$N_2 = m_1 m_2 \cdots m_d \text{ where } m_i < \frac{0.1 \log n}{L_2 \log p} < m_i m_j \text{ for all } i, j \in [d]\,(i \neq j). \tag{2}$$

For each $i \in [d]$, define $t_i = \lfloor \frac{0.1 \log n}{m_i L_2 \log p} \rfloor \geq 1$. In the following, we will pack t_i instances of degree-m_i DFTs over $\mathbb{F}_{p^{L_2}}$ in a machine word.

Lemma B.6. *For each $i \in [d]$, after $n^{0.2+o(1)}$ time pre-processing, computing t_i instances of degree-m_i DFTs over $\mathbb{F}_{p^{L_2}}$ can be done in $O(1)$ time (assuming a compactly represented input and output).*

Similarly, this also holds for the task of multiplying degree-m_i polynomials.

Proof. The number of such t_i instances of degree-m_i polynomials over $\mathbb{F}_{p^{L_2}}$ is $((p^{L_2})^{m_i})^{t_i} \leq n^{0.1}$ by the definition of t_i, so we can preprocess a look-up table in $n^{0.1+o(1)}$ time, which later allows us to look up the DFT answers in $O(1)$ time in word RAM with $\Theta(\log n)$-bit words (assuming the inputs and outputs are packed into $O(1)$ words). A similar argument applies to the task of computing t_i instances of degree-m_i polynomial multiplication, which takes preprocessing time $n^{0.2+o(1)}$. □

Now we describe our second-level algorithm.

Lemma B.7 (Second level)**.** *After $n^{0.2+o(1)}$ time pre-processing, multiplying two degree-L_1 polynomials over $\mathbb{F}_p$ can be done in $O((L_1 \log L_1) \cdot \frac{\log p}{\log n})$ time.*

Proof. Assume $p \leq L_1^{O(1)}$; otherwise use Lemma B.5 instead. Recall $L_1' \in [L_1, 2L_1]$ and $L_2 = L_1'/N_2 = \Theta(L_1/N_2)$. Hence, we first reduce the task of multiplying two degree-L_1 polynomials over $\mathbb{F}_p$ to $O(1)$ instances of multiplications of two degree-$\lfloor (N_2 - 1)/2 \rfloor$ polynomials over $\mathbb{F}_{p^{L_2}}$. In more details, this is achieved by packing contiguous $\lfloor L_2/2 \rfloor$ coefficients from $\mathbb{F}_p$ into an element in $\mathbb{F}_{p^{L_2}}$ (where we divided by two so that the products will not overflow modulo the irreducible monic polynomial of degree L_2). This way, the problem becomes the multiplication of two polynomials over $\mathbb{F}_{p^{L_2}}$ of degree $L_1/\lfloor L_2/2 \rfloor = O(N_2)$, which can be easily reduced to $O(1)$ instances of multiplications of two degree-$\lfloor (N_2 - 1)/2 \rfloor$ polynomials over $\mathbb{F}_{p^{L_2}}$.

In the following, we describe how to perform this multiplication, whose product should be a polynomial over $\mathbb{F}_{p^{L_2}}$ of degree at most $N_2 - 1$.

Recall N_2 has a smooth factorization $N_2 = \prod_{i=1}^d m_i$ given by Eq. (2), and recall that we computed a primitive N_2-th root of unity $\omega_{N_2} \in \mathbb{F}_{p^{L_2}}$. Hence we can use the standard Cooley-Tukey FFT algorithm of length N_2 to do the multiplication (see e.g., [HvdHL17, Section 2.3]). In the following, we first recall the DFT procedure, and later describe the implementation details in word RAM.

The DFT algorithm. Given input array $(a_0, \ldots, a_{N_2-1}) \in (\mathbb{F}_{p^{L_2}})^{N_2}$, we initialize the working array $A := (a_{rev(0)}, \ldots, a_{rev(N_2-1)})$ where $rev(\cdot)$ is a permutation defined as follows (analogous to the bit-reversal permutation used in the radix-2 version): if $x = \sum_{i=1}^d x_i \cdot m_1 m_2 \cdots m_{i-1}$ (where $0 \leq x_i < m_i$), then $rev(x) := \sum_{i=1}^d x_i \cdot m_{i+1} \ldots m_{d-1} m_d$. Then we perform d rounds of computation on the working array A, where in the i-th round $(1 \leq i \leq d)$ we perform the following operations (denote $M_i = m_1 m_2 \cdots m_i$):

1. For each $0 \leq k < N_2/M_i$ and $0 \leq j < M_{i-1}$, let $l = kM_i + j$, and for all $0 \leq s < m_i$, multiply the "twiddle factors":
$$A[l + sM_{i-1}] \leftarrow A[l + sM_{i-1}] \cdot \omega_{M_i}^{sj}.$$
In total there are N_2 scalar multiplications over $\mathbb{F}_{p^{L_2}}$ in this round.

2. For each $0 \le k < N_2/M_i$ and $0 \le j < M_{i-1}$, let $l = kM_i + j$, and perform a length-m_i in-place DFT:

$$(A[l], A[l+M_{i-1}], , \ldots, A[l+(m_i-1)M_{i-1}]) \leftarrow DFT(A[l], A[l+M_{i-1}], , \ldots, A[l+(m_i-1)M_{i-1}]).$$

In total there are N_2/m_i instances of length-m_i DFT over $\mathbb{F}_{p^{L_2}}$ in this round.

One can verify that after d rounds, the working array A becomes the correct DFT result, i.e., $A[k] = \sum_{j=0}^{N_2-1} a_j \cdot \omega_{N_2}^{jk}$.

Implementation of DFT. To implement the DFT algorithm described above, we always use a compact representation of the working array $A \in (\mathbb{F}_{p^{L_2}})^{N_2}$ into $O(\frac{N_2 L_2 \log p}{\log n})$ words, and we need to use bit parallelism to speed up these operations.

- Item 1 (multiplying the "twiddle factors"):

  In constant time, we multiply the twiddle factors to $\Theta(\frac{\log n}{L_2 \log p})$ contiguous elements (represented in $O(1)$ words) in the working array A using table look-up (similar to Lemma B.6 with m_i set to 1). (Note that $\frac{\log n}{L_2 \log p} \le N_2$.) In order to do this table look-up, we also need to prepare a compact representation of the $\Theta(\frac{\log n}{L_2 \log p})$ twiddle factors applied to the working array. Note that these twiddle factors are fixed in the algorithm and do not depend on the input, so we can pre-compute the compact representations of them in $\mathrm{poly}(N_2 \cdot L_2 \log p) \le n^{o(1)}$ time.

  The total time for Item 1 over all d rounds (ignoring preprocessing) is $O(d \cdot (N_2 \cdot L_2 \log p)/\log n)$.

- Item 2 (length-m_i DFTs):

  In the i-th round, we need to apply length-m_i DFTs on the working array, and we want to speed them up by using Lemma B.6 to perform t_i DFTs in a batch in constant time. To do this, we need to first collect the array elements $A[l], A[l + M_{i-1}], \ldots, A[l + (m_i - 1)M_{i-1}]$ participating in each DFT into a contiguous range of memory in compact representation. (Note that we only need to do this when $i \ge 2$; for $i = 1$, since $M_{i-1} = 1$, these elements are already in a contiguous range.) More specifically, we need to permute array A into A' so that

  $$A'[kM_i+jm_i+s] = A[kM_i+j+sM_{i-1}] \text{ for all } 0 \le k < N_2/M_i, 0 \le j < M_{i-1}, 0 \le s < m_i. \quad (3)$$

  In other words, if we view the length-N_2 working array A as N_2/M_i chunks each representing an $m_i \times M_{i-1}$ matrix in row-major order, then A' is obtained by transposing these matrices into column-major order. After permuting A into A', we can perform the required DFTs on A' with time complexity linear in the number of words using the look-up tables from Lemma B.6, and then we permutate them back by running the transposition step in reverse. Note that the running time for performing DFTs on A' is dominated by the transposition steps.

  Transposing an $m_i \times M_{i-1}$ matrix can be done by a divide-and-conquer algorithm (similar to [Tho02, Lemma 9]) with recursion depth $\log(m_i)$: we start with m_i length-M_{i-1} lists each corresponding to a leaf of the recursion tree, and at each internal node of the recursion tree we interleave the lists returned by its two child nodes. Here, using word operations (which can be replaced by table look-ups after preprocessing), interleaving two lists can be done with time complexity linear in the number of words in their compact representations. Hence, transposing an $m_i \times M_{i-1}$ matrix (with entries from $\mathbb{F}_{p^{L_2}}$) via divide-and-conquer takes total time $\sum_{q=0}^{\log m_i} 2^q \cdot (O(\frac{(m_i/2^q)M_{i-1}L_2 \log p}{\log n}) +$

$O(1)) \le O((\log m_i) m_i M_{i-1} L_2 \frac{\log p}{\log n} + m_i)$, and transposing N_2/M_i such matrices in total takes $O((\log m_i) N_2 L_2 \frac{\log p}{\log n} + N_2/M_{i-1})$ time. For $i \ge 3$, we have $M_{i-1} \ge m_1 m_2 > \frac{0.1 \log n}{L_2 \log p}$ from Eq. (2), and the second term N_2/M_{i-1} in the time complexity is dominated, so the run time becomes $O((\log m_i) N_2 L_2 \frac{\log p}{\log n})$. For the remaining case $i = 2$, the same run time can be achieved by slightly modifying the divide-and-conquer matrix transposition algorithm: when the total bit length of the lists in the current recursion subtree is below $0.1 \log n$, we simply look up the transposition result from a preprocessed table instead of recursing.

To summarize, the total time for Item 2 for all d rounds is

$$
\begin{aligned}
& O\left(\sum_{i=1}^{d} \log(m_i) \cdot N_2 L_2 \frac{\log p}{\log n}\right) \\
&= O(\log N_2) \cdot N_2 L_2 \frac{\log p}{\log n} && \text{(by Eq. (2))} \\
&= O(\log N_2) \cdot L_1 \frac{\log p}{\log n} && \text{(by } L_2 = L_1'/N_2 \text{ and } L_1' = \Theta(L_1)\text{)} \\
&= O\left(L_1 \log L_1 \cdot \frac{\log p}{\log n}\right).
\end{aligned}
$$

Finally, note that the initialization step (applying the $rev(\cdot)$ permutation to the input array) can be done in a similar fashion to the transposition steps described in Item 2, with the same total time complexity $O\left(L_1 \log L_1 \cdot \frac{\log p}{\log n}\right)$.

Note that the total time of Item 1 is dominated by Item 2, so the total time complexity of the algorithm is $O\left(L_1 \log L_1 \cdot \frac{\log p}{\log n}\right)$. The total pre-processing time of calling Lemma B.6 d times is $O(n^{0.2+o(1)} \cdot d) = n^{0.2+o(1)}$, and the pre-processing time for other look-up tables used by the algorithm can also be bounded similarly by $n^{0.2+o(1)}$. □

The proof of Lemma B.7 described above can also prove the following slightly stronger statement:

Corollary B.8. *Let $\tilde{L}_1 \in [L_1^{0.2}, L_1^{10}]$ be a power of two. After $n^{0.2+o(1)}$ time pre-processing, multiplying two degree-$\tilde{L}_1$ polynomials over $\mathbb{F}_p$ can be done in $O((\tilde{L}_1 \log \tilde{L}_1) \cdot \frac{\log p}{\log n})$ time.*

Proof. The only bounds on L_1 that we used in proving Lemma B.7 are $L_1 \in (\log n)^{\Theta(\log \log \log n)}$ and $p \le L_1^{O(1)}$, which also hold for $\tilde{L}_1$. Hence we can simply repeat the proof of Lemma B.7 with $\tilde{L}_1$ in place of L_1. □

Finally we describe the first level of our algorithm, proving Lemma 2.2.

Proof of Lemma 2.2. Recall $p \le n^{0.01}$, $L_1 = n'/N_1$ and $n = \Theta(n')$. By the same reasoning as in the proof of Lemma B.7, we can reduce the task of multiplying two degree-n polynomials over $\mathbb{F}_p$ to $O(1)$ instances of polynomial multiplication over $\mathbb{F}_{p^{L_1}}$ whose product has degree at most $N_1 - 1$. Note that we can find a representation of $\mathbb{F}_{p^{L_1}}$ by finding an irreducible monic polynomial of degree L_1, which can be done in $\widetilde{O}(L_1^4 p^{1/2}) = n^{0.005+o(1)}$ time deterministically [Sho88]. Since we have shown earlier that $N_1 \mid p^{L_1} - 1$, we can find a primitive N_1-th root of unity $\omega_{N_1} \in \mathbb{F}_{p^{L_1}}$ in $\widetilde{O}(L_1^9 p) = n^{o(1)} \cdot n^{0.01}$ time [HvdHL17, Lemma 3.3]. Let $N_1 = \prod_{i=1}^{d} m_i$ as in Corollary B.4, where $\sqrt{L_1}/2 \le m_i \le L_1^3$.

To do this multiplication, we run Cooley-Tukey FFT using this smooth N_1-th root. Similar as the proof of Lemma B.7, it involves d rounds of computation on a (compactly represented) working array of N_1 elements from $\mathbb{F}_{p^{L_1}}$, where the i-th round involves the following operations.

1. N_1 scalar multiplications over $\mathbb{F}_{p^{L_1}}$: multiply the "twiddle factors" to each of the N_1 elements in the array.

   The cost for preparing all possible twiddle factors $\{\omega_{N_1}^j\}_{j\in[N_1]}$ is N_1 scalar multiplications over $\mathbb{F}_{p^{L_1}}$, which can be absorbed into the cost of this step. (In contrast to the proof of Lemma B.7, here we do not need to prepare the compact representations of multiple twiddle factors packed into one word, since here each twiddle factor already occupies more than one word.)

2. N_1/m_i instances of length-m_i DFT over $\mathbb{F}_{p^{L_1}}$.

Let $T_{D,L_1}(\ell)$ denote the cost of computing the DFT of a length-ℓ polynomial over $\mathbb{F}_{p^{L_1}}$, and let T_{M,L_1} denote the cost of scalar multiplication over $\mathbb{F}_{p^{L_1}}$. Then the total time complexity for FFT is (up to constant factors)

$$\sum_{i=1}^{d}\left(N_1\cdot T_{M,L_1}+\frac{N_1}{m_i}T_{D,L_1}(m_i)\right).$$

Now we analyze the two terms separately.

- To analyze T_{M,L_1}, note that a scalar multiplication over $\mathbb{F}_{p^{L_1}}$ can be done by computing the product of two degree-L_1 polynomials over $\mathbb{F}_p$, and then mapping it back to $\mathbb{F}_{p^{L_1}}$ by reducing modulo a degree-L_1 monic irreducible polynomial over $\mathbb{F}_p$. By Lemma B.7, multiplying two degree-L_1 polynomials over $\mathbb{F}_p$ can be done in $O((L_1\log L_1)\frac{\log p}{\log n})$ time. Using Newton's iteration (see e.g., [vzGG13, Section 9]), degree-L_1 polynomial division with remainder can be reduced to $O(\log L_1)$ instances of polynomial multiplication with degrees $L_1, \frac{L_1}{2}, \frac{L_1}{4}, \frac{L_1}{8}, \ldots$ respectively. For multiplication with degree $\geq L_1^{0.2}$, we invoke Corollary B.8. For smaller degree, we use brute-force quadratic-time multiplication. The total time for degree-L_1 polynomial division is thus (up to a constant factor)

  $$\sum_{j=0.2\log_2 L_1}^{\log_2 L_1}(2^j\log 2^j)\frac{\log p}{\log n}+\sum_{j=0}^{0.2\log_2 L_1}(2^j)^2\leq O(L_1\log L_1)\cdot\frac{\log p}{\log n}+O(L_1^{0.4})=O(L_1\log L_1)\cdot\frac{\log p}{\log n}.$$

  Hence, $T_{M,L_1}=O((L_1\log L_1)\frac{\log p}{\log n})$.

- To analyze $T_{D,L_1}(m_i)$, we use Bluestein's chirp transform (see [HvdHL17, Section 2.5]) to reduce the task of computing a length-m_i DFT over $\mathbb{F}_{p^{L_1}}$ to multiplying two degree-m_i polynomials over $\mathbb{F}_{p^{L_1}}$. This can further be reduced to multiplying degree-$2m_iL_1$ polynomials over $\mathbb{F}_p$ via Kronecker substitution (see [HvdHL17, Section 2.6]), which can be solved using Corollary B.8 (recall $m_i\leq L_1^3$) in time $O(m_iL_1\cdot\log(m_iL_1)\cdot\frac{\log p}{\log n})$. Afterwards, we divide m_i degree-$2L_1$ polynomials over $\mathbb{F}_p$ by the degree-L_1 irreducible monic polynomial over $\mathbb{F}_p$, to map the elements back to $\mathbb{F}_{p^{L_1}}$, in total time $O(m_iL_1\cdot\log(L_1)\cdot\log p/\log n)$ (similar to the previous paragraph). Hence, $T_{D,L_1}(m_i)\leq O(m_iL_1\log(L_1)\cdot\frac{\log p}{\log n})$

Hence, the total time becomes (up to constant factors)

$$\begin{aligned}
&\sum_{i=1}^{d} \big(N_1 \cdot T_{M,L_1} + \frac{N_1}{m_i} T_{D,L_1}(m_i)\big) \\
&\leq \sum_{i=1}^{d} \big(N_1 (L_1 \log L_1) \frac{\log p}{\log n} + \frac{N_1}{m_i} m_i L_1 \log(L_1) \cdot \frac{\log p}{\log n}\big) \\
&\leq O\big(d N_1 (L_1 \log L_1) \frac{\log p}{\log n}\big) \\
&\leq O\big(d \cdot n \log L_1 \cdot \frac{\log p}{\log n}\big).
\end{aligned}$$

Recall that Corollary B.4 gave the factorization $N_1 = \prod_{i=1}^{d} m_i$ with $m_i \in L_1^{\Theta(1)}$, so $d \log L_1 = \Theta(\log N_1) = \Theta(\log n)$, and the final run time becomes $O(n \log p)$ as desired. □