Output-Sparse Matrix Multiplication Using Compressed Sensing

Huck Bennett
University of Colorado Boulder, CO, USA

Karthik Gajulapalli
Georgetown University, Washington, DC, USA

Alexander Golovnev
Georgetown University, Washington, DC, USA

Evelyn Warton
Oregon State University, Corvallis, OR, USA

Abstract

We give two algorithms for *output-sparse matrix multiplication* (OSMM), the problem of multiplying two $n \times n$ matrices A, B when their product AB is promised to have at most $O(n^\delta)$ many non-zero entries for a given value $\delta \in [0, 2]$. We then show how to speed up these algorithms in the *fully sparse matrix multiplication* (FSMM) setting, where the input matrices A, B are themselves sparse. All of our algorithms work over arbitrary rings.

Our first, deterministic algorithm for OSMM works via a two-pass reduction to compressed sensing. It runs in roughly $n^{\omega(\delta/2,1,1)}$ time, where $\omega(\cdot,\cdot,\cdot)$ is the rectangular matrix multiplication exponent. This substantially improves on prior deterministic algorithms for output-sparse matrix multiplication.

Our second, randomized algorithm for OSMM works via a reduction to compressed sensing and a variant of matrix multiplication verification, and runs in roughly $n^{\omega(\delta-1,1,1)}$ time. This algorithm and its extension to the fully sparse setting have running times that match those of the (randomized) algorithms for OSMM and FSMM, respectively, in recent work of Abboud, Bringmann, Fischer, and Künnemann (SODA, 2024). Our algorithm is quite simple when taking a suitable compressed sensing scheme as a black box.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Pseudorandomness and derandomization; Theory of computation → Expander graphs and randomness extractors; Theory of computation → Error-correcting codes

Keywords and phrases Matrix Multiplication, Sparse Matrices, Compressed Sensing

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.40

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/abs/2508.10250` [6]

Funding *Huck Bennett*: Supported by NSF Grant CCF-2432132.
Karthik Gajulapalli: Supported by NSF grant CCF-2338730.
Alexander Golovnev: Supported by NSF grant CCF-2338730.

Acknowledgements HB and AG would like to thank Divesh Aggarwal, the National University of Singapore (NUS), and the Centre for Quantum Technologies (CQT). Some of this work was completed while visiting them. The authors would also like to thank Thatchaphol Saranurak for suggesting the use of the expander construction in [17], and Noah Stephens-Davidowitz for helpful comments. Finally, the authors would like to thank anonymous reviewers for providing helpful references and comments (in particular, for noting the connection between [1, Lemma 3.1] and our work).

1 Introduction

The Matrix Multiplication Problem (MM) is to compute the product AB of two $n \times n$ matrices A, B given as input. It is one of the most fundamental problems in computer science. The running time $O(n^\omega)$ of the fastest algorithm for MM is governed by the *matrix multiplication exponent* $\omega \leq 2.372$ (see [2] for the current best bound), and it is a major open question whether $\omega = 2$.

While one line of work has sought to improve ω, another has tried to find faster algorithms for important special cases of MM. In this work, we give new algorithms for the *Output-Sparse Matrix Multiplication Problem* (OSMM), where the product AB of the input matrices A and B is promised to be *sparse*. Specifically, we consider MM when AB is promised to have $O(n^\delta)$ many non-zero entries for a parameter $\delta \in [0, 2]$. We denote this problem for matrices over a ring R by δ-OSMM$_R$. We also discuss how to extend our algorithms to the *fully sparse* setting, in which the input matrices A and B are themselves sparse. This setting was recently studied in depth in seminal work of Abboud, Bringmann, Fischer, and Künnemann [1].

1.1 Our Work

Our main results are a deterministic algorithm and a randomized algorithm for OSMM over arbitrary rings R including, in particular, finite fields, the integers, and the real numbers. The algorithms perform a similar number of ring operations regardless of R.

Define the *rectangular matrix multiplication exponent* $\omega(\alpha, \beta, \gamma)$ to be the infimum over values $\omega' \geq 0$ such that there is an algorithm for multiplying an $n^\alpha \times n^\beta$ matrix and an $n^\beta \times n^\gamma$ matrix in $O(n^{\omega'})$ time. Define the *dual matrix multiplication exponent* $\omega^\perp$ to be the supremum over values $\omega' \geq 0$ such that $\omega(1, 1, \omega') = 2$.[1]

The theorems corresponding to our main two OSMM algorithms are as follows. Our first algorithm dramatically improves on the state of the art for deterministic algorithms; see the comparison with prior work in Figure 1 and Section 1.2.

▶ **Theorem 1** (Deterministic OSMM). *Let $\delta \in [0, 2]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a deterministic algorithm for solving δ-OSMM$_R$ on $n \times n$ matrices that performs $O(n^{\omega(\delta/2,1,1)+\varepsilon})$ operations over R.*

Our second, randomized algorithm, presented in Theorem 2, has running time that matches an algorithm from [1] using a reduction to a variant of the Matrix Multiplication Verification Problem (MMV). See Section 1.2 for a comparison of our Theorem 2 and the algorithm in [1]. We show that the running times of these algorithms are optimal in Section 4.3.

▶ **Theorem 2** (Randomized OSMM). *Let $\delta \in [0, 2]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a randomized algorithm for solving δ-OSMM$_R$ on $n \times n$ matrices that performs $O(n^{\beta+\varepsilon})$ operations over R, where*

$$\beta := \begin{cases} 2 & \text{if } \delta \leq 1 + \omega^\perp \ , \\ \omega(\delta - 1, 1, 1) & \text{otherwise} \ . \end{cases}$$

[1] The dual matrix multiplication exponent is often denoted by α in the literature.

Since $\omega^{\perp} \geq 0.321$ [35], the algorithms in Theorems 1 and 2 run in essentially optimal n^2 time for all $\delta \leq 0.642$ and $\delta \leq 1.321$, respectively. Furthermore, both algorithms run in $O(n^{\omega-\varepsilon})$ time for some $\varepsilon = \varepsilon(\delta) > 0$ for every $\delta < 2$.[2] That is, both algorithms are faster than the fastest known algorithms for matrix multiplication as long as AB is promised to be even, say, $n^{1.99}$-sparse. We also extend our algorithms to the *fully sparse* setting where the input matrices A, B are themselves sparse (in addition to having the promise that their product AB is sparse), and get faster algorithms in this case; see Section 4 and Theorems 18 and 19. Although we choose to highlight our somewhat simpler OSMM algorithms, they actually follow as corollaries from our algorithms for the fully sparse case.

We note that our extension of Theorem 2 to the fully sparse setting (given in Theorem 19) is equivalent to [1, Lemma 3.1], the main technical lemma in [1] for fully sparse matrix multiplication on general rings. However, our corresponding algorithm uses different techniques and is arguably simpler. In Proposition 20 we also show that our randomized algorithms in Theorems 2 and 19 (and therefore also [1, Lemma 3.1]) are *optimal* via a reduction from rectangular matrix multiplication.

We also provide an exposition of a nearly optimal, deterministic compressed sensing that works over any ring in an appendix of the full version of this paper [6]. The scheme instantiates the compressed sensing scheme from [7] with the expander construction from [17]. Although likely known as folklore, this scheme does not seem to have been analyzed in detail before.

1.2 Applications and Comparison with Prior Work

Input- and output-sparse matrix multiplication have a number of applications including answering certain database queries [4, 34, 10], computing transitive closures in sparse graphs [34, 8], multi-source breadth-first search [37, 16, 13], Markov clustering [33], and error correction of matrix products [14, 23, 32].[3]

A long line of work has studied both input- and output-sparse matrix multiplication [18, 36, 20, 28, 4, 30, 24, 21, 34, 23, 32, 10, 1]. Our deterministic algorithm in Theorem 1 substantially improves on prior deterministic δ-OSMM algorithms for most values of δ, and is never slower than them. See Figure 1. As noted above, our algorithm runs in essentially optimal quadratic time for $\delta \leq 0.642$, and for every constant $\delta < 2$ it runs in $O(n^{\omega-\varepsilon})$ time for some $\varepsilon = \varepsilon(\delta) > 0$.

The main prior works on deterministic OSMM are Kutzkov [24], which gives a $O(n^2 + n^{2\delta+1})$-time algorithm for OSMM over the reals, and Künnemann [23], which gives a $O(n^{2+\delta/2} + n^{2\delta})$-time algorithm for OSMM over the integers. Their algorithms only run in $O(n^{\omega-\varepsilon})$ time for $\delta < (\omega-1)/2 \approx 0.686$ and $\delta < 2(\omega-2) \approx 0.744$, respectively. (One can also show analytically that Theorem 1 is at least as fast as [24, 23] for all δ.[4]) Moreover, if one is interested in only using "combinatorial algorithms" for matrix multiplication (i.e., assuming that $m \times n \times p$ rectangular MM takes $O(mnp)$ time), our deterministic algorithm runs in $\widetilde{O}(n^{2+\delta/2})$ time. So, our algorithm is strictly faster than the algorithms in [24, 23] for $\delta > 4/3$.

[2] Note that $\delta - 1 \leq \delta/2$ for $\delta \in [0, 2]$ so the randomized algorithm in Theorem 2 is always at least as fast as the deterministic algorithm in Theorem 1.

[3] We note that error correction of matrix products is equivalent to OSMM.

[4] By Theorem 7 we have that $\omega(1, 1, \delta/2) \leq 1.825 + 0.548\delta/2$ from which it follows that $\omega(1, 1, \delta/2) \leq 2\delta + 1$ for all $\delta \geq 0.5$. So, our algorithm is always at least as fast as the algorithm in [24]. Furthermore, by Proposition 6, Item 1, $\omega(\delta/2, 1, 1) \leq 2 + \delta/2$ and so our algorithm is always at least as fast as the algorithm in [23].

Our randomized OSMM algorithm in Theorem 2 and the breakthrough randomized algorithm in [1] both run in roughly $n^{\omega(\delta-1,1,1)}$ time. They work in a similar way at a very high level, however there are notable differences. Perhaps more starkly, unlike [1], our randomized algorithm works by reducing OSMM to a variant of the Matrix Multiplication Verification (MMV) problem. Moreover, solving this variant of MMV is the algorithm's *only* use of randomness.

The algorithms in Theorem 2 and [1] both work by (implicitly) partitioning the output matrix AB into blocks of rows or columns with similar sparsity, "densifying" corresponding blocks in the input matrices, multiplying these dense blocks, and then recovering the (sparse) output matrix. To do this, [1] uses hash functions for the densification and a method for computing a small overapproximation of the support of the output matrix. On the other hand, our randomized OSMM algorithm, given in Algorithm 2, works directly with the primitives of a compressed sensing scheme, and it is simple to describe and analyze when these are taken as a black box. In particular, our algorithm performs densification by multiplying by a measurement matrix, and recovers (parts of) the output matrix via sparse recovery. (One can also interpret the hashing-based densification method in [1] as multiplying by a certain (random) measurement matrix. Our algorithm and analysis abstract this out.) See Section 1.3 for a detailed technical overview of our algorithms.

We also note a connection between our algorithm for OSMM and an algorithm from [5] for the All Zeroes Problem, the problem of checking whether $AB = 0$ for two $n \times n$ input matrices A, B. (We note that [23] showed that the All Zeroes Problem and the better-known MMV problem are essentially equivalent under a nearly sparsity-preserving reduction.) Specifically, Theorem 1 achieves the same running time of roughly $n^{\omega(\delta/2,1,1)}$ for δ-OSMM that a deterministic algorithm in [5] achieved for the All Zeroes Problem with the same promise that the input matrices A, B satisfy $\|AB\|_0 \leq O(n^\delta)$. The All Zeroes Problem with the guarantee $\|AB\|_0 \leq O(n^\delta)$ reduces to δ-OSMM (this is a trivial decision-to-search reduction), and so one can view Theorem 1 as an upgrade of the deterministic algorithm in [5]. We also emphasize again that Theorems 1 and 2 work over *any* ring R whereas the algorithms from each of [24, 23, 5] only work on specific domains.

The sole deterministic OSMM algorithm that is faster than ours is an algorithm that (only) works for multiplication over nonnegative integers in [1, Lemma 3.11].[5] Their algorithm's running time is stated for the fully sparse setting, but it runs in roughly $O(n^{\omega(\delta-1,1,1)})$ time in the output sparse setting. This matches the running time of randomized algorithms in their work and ours for OSMM over general rings.

1.3 Technical Overview

We next give an overview of our algorithms, which use simple primitives: fast rectangular matrix multiplication, a compressed sensing scheme with good parameters, and an algorithm for a variant of matrix multiplication verification. Define $\|\boldsymbol{v}\|_0$ (respectively, $\|M\|_0$) to be the number of non-zero entries in (i.e., Hamming weight of) a vector $\boldsymbol{v}$ (respectively, matrix M). We call a vector $\boldsymbol{v}$ (respectively, matrix M) *t-sparse* if $\|\boldsymbol{v}\|_0 \leq t$ (respectively, if $\|M\|_0 \leq t$).

[5] We note that algorithms for matrix multiplication over nonnegative integers imply algorithms for Boolean matrix multiplication as a special case. Recall that the (i, j)th entry of the Boolean product of matrices $A, B \in \{0, 1\}^{n\times n}$ is defined as $\vee_{k=1}^{n}(A_{i,k} \wedge B_{k,j})$. To compute this product it suffices to compute the standard matrix product AB over the integers and replace all positive entries of the output with ones.

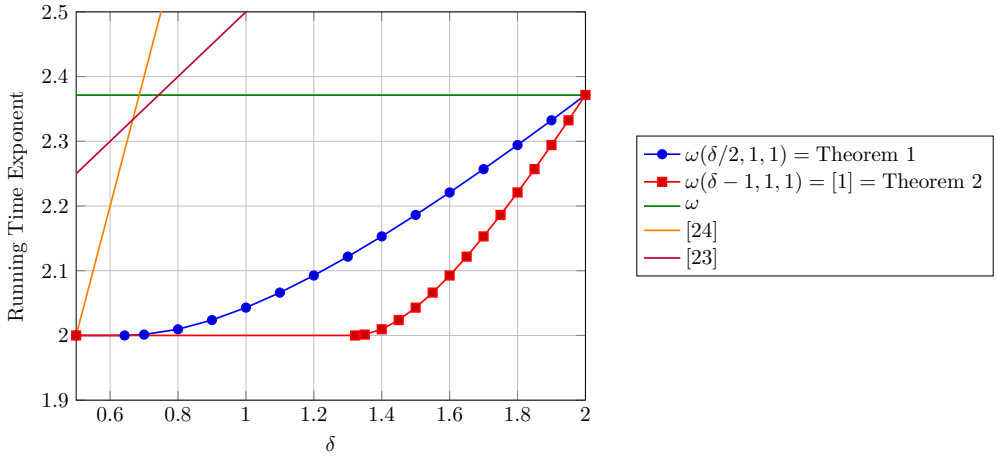

Figure 1 A plot of the running time exponents of several algorithms for δ-OSMM, with arbitrarily small polynomial factors suppressed. The blue curve gives the running time of our deterministic algorithm from Theorem 1, which runs in $n^{\omega(\delta/2,1,1)}$ time. The red curve gives the running time of the randomized algorithm from [1] and our randomized algorithm from Theorem 2. These algorithms both run in roughly $n^{\omega(\delta-1,1,1)}$ time. The points on the curves come from upper bounds on rectangular MM exponents in [35], and the line segments connecting them are justified by the convexity of $\omega(\cdot, 1, 1)$. We also plot the running time exponents of the $O(n^{\max\{2,2\delta+1\}})$-time algorithm of Kutzkov [24] and the $O(n^{\max\{2+\delta/2,2\delta\}})$-time algorithm of Künnemann [23] – which are the main prior works giving *deterministic* algorithms for OSMM – as well as the MM exponent ω.

1.3.1 Output-column-sparse MM using compressed sensing

In the *compressed sensing* problem with parameters $n, m, t \in \mathbb{Z}^+$ with $1 \le t \le n$, the goal is to compute an $m \times n$ matrix $H = H(m, n, t)$ called a *measurement matrix*, and to design an efficient *sparse recovery* algorithm $\mathcal{R}$ (depending on H) such that for every t-sparse vector $\boldsymbol{x}$ of length n, $\mathcal{R}(H\boldsymbol{x}) = \boldsymbol{x}$. That is, the goal is for $\mathcal{R}$ to be able to recover any t-sparse vector $\boldsymbol{x}$ from $H\boldsymbol{x}$.

Several previous works including [20] use compressed sensing to solve MM in the case where every column $\boldsymbol{c}_i$ of the output matrix $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) := AB$ is promised to be t-sparse. The technique works by computing HAB parenthesized as $(HA)B$ using fast rectangular matrix multiplication, and then recovering the columns $\boldsymbol{c}_i$ of $C = AB$ from HC using $\mathcal{R}$ as $\boldsymbol{c}_i = \mathcal{R}(H\boldsymbol{c}_i)$ for $i = 1, \ldots, n$. If computing H takes $T_{\mathcal{M}}(n)$ time, $\mathcal{R}$ takes $T_{\mathcal{R}}(n)$ time, and $m = O(n^\beta)$ for some $\beta \in [0, 1]$, then for any constant $\varepsilon > 0$ the overall running time of this algorithm is

$$O(T_{\mathcal{M}}(n) + n^{\omega(\beta,1,1)+\varepsilon} + n \cdot T_{\mathcal{R}}(n)) \ . \tag{1}$$

We note that [20] uses the compressed sensing scheme of [19]. The compressed sensing scheme there has good parameters and has a fast sparse recovery algorithm $\mathcal{R}$. However, while [19] uses a polynomial-time construction of expanders, it is unclear that its construction runs in better-than-n^ω time. (We achieve $T_{\mathcal{M}}(n) = \widetilde{O}(n)$ time in Theorem 13.)

1.3.2 Reducing OSMM to compressed sensing

As our first contribution, we show how to replace the assumption in the above compressed-sensing-based algorithm for MM that every column of $C = AB$ is t-sparse with the "global" assumption that C is t^2-sparse. If $t^2 = O(n^\delta)$, this is exactly the assumption in δ-OSMM.

To do this, we use a two-pass reduction, which roughly speaking first computes a matrix C' that agrees with C on sparse columns and then computes all rows of $C - C'$. This reduction appears formally as Algorithm 1.

Assume without loss of generality that $\mathcal{R}$ always outputs *some* vector of length n even when its input is not t-sparse. More specifically, assume that $\boldsymbol{c}'_i := \mathcal{R}(H\boldsymbol{c}_i)$ is always a vector of length n with the guarantee that $\boldsymbol{c}'_i = \boldsymbol{c}_i$ if $\boldsymbol{c}_i$ is t-sparse. Let $C' := (\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_n)$. The key observation is that if $C = AB$ is t^2-sparse, then *every* row of $C - C'$ must be t-sparse.[6] (Similar observations were previously used, e.g., in the proof of [14, Theorem 4.3] and in [5].) To see this, note that if C is t^2-sparse, then it has fewer than t columns that are *not* t-sparse, and that these are the only (potentially) non-zero columns in $C - C'$. We can then recover the entries of $(C - C')^T$ (and therefore also C, since we know C') by computing $H(C - C')^T$ as $(HB^T)A^T - H(C')^T$ and then running $\mathcal{R}$ on each column of the result.

1.3.3 Reducing OSMM to compressed sensing and a variant of MMV

Our second reduction is from OSMM to compressed sensing and a variant of the matrix multiplication verification problem (MMV) that we call Column-Wise-MMV. The goal of Column-Wise-MMV is, given three matrices A, B, and C' as input, to identify all of the indices $i \in [n]$ such that $\boldsymbol{c}_i \neq \boldsymbol{c}'_i$, where $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) := AB$ and $C' = (\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_n)$. This reduction appears formally as Algorithm 2.

The main idea of our second reduction is to sequentially "guess and check" all columns of C of sparsity at most 2^i for $i = 0, \ldots, \lceil \log n \rceil$. Namely, in iteration i we use a compressed sensing scheme as above to compute a matrix C' that agrees with all columns of C of sparsity at most $t = 2^i$. (As before, we assume that $\mathcal{R}$ always outputs some vector of length n regardless of its input.) We then use our Column-Wise-MMV oracle to identify which of the columns of C were computed correctly in C'. We use this information to update the corresponding columns of our output matrix and to remove the corresponding columns from B. We emphasize that this reduction itself is deterministic, and the only use of randomness in the algorithm summarized in Theorem 2 comes from solving Column-Wise-MMV.

Correctness follows simply by analyzing the last iteration (in which $i = \lceil \log n \rceil$) since each column of C has sparsity at most $n \leq 2^{\lceil \log n \rceil}$. In fact, the reduction does not require an upper bound on the sparsity of C as input and works regardless of this sparsity – it is a reduction from MM, not just OSMM. However, the sparser C is the faster the reduction is. Let $J_{i+1} \subseteq [n]$ be the set of indices of columns of C with more than 2^i non-zero entries.[7]

Assume that it is possible to multiply the $m \times n$ measurement matrix H by an arbitrary $n \times n$ matrix A in $O(n^{2+\varepsilon})$ time for any constant $\varepsilon > 0$. When we instantiate the reduction with the concrete compressed sensing scheme described in Theorem 13 this will be the case since our measurement matrices H will be sparse. Then the bottleneck in each iteration i is computing the product of HA (an $m \times n$ matrix) and B (an $n \times |J_i|$ matrix). Let $\alpha := i/\log n$. We know that $|J_{i+1}| \leq O(n^\delta/2^i) = O(n^{\delta-\alpha})$, and if $m \approx t = 2^i = n^\alpha$ then it is possible to compute the product of HA and B in roughly $n^{\omega(\alpha,1,\delta-\alpha)}$ time. By Corollary 8,

[6] Note that it is *not* the case that every non-zero entry of C must be contained in a t-sparse row or column. For example, if $C = (\boldsymbol{e}_1 \cdot \mathbf{1}^T) + (\mathbf{1} \cdot \boldsymbol{e}_1) \in \mathbb{Z}^{n \times n}$ then $C_{1,1} = 2$ and both the first column and first row of C have n non-zero entries. (Here $\boldsymbol{e}_1$ and $\mathbf{1}$ denote the first standard normal basis vector and the all-ones vector, respectively.) Yet, C is $O(n)$-sparse.

[7] In Algorithm 2, J_{i+1} can technically be a subset of these columns if the sparse recovery algorithm happens to succeed (i.e., when $\mathcal{R}(H\boldsymbol{x}) = \boldsymbol{x}$) at an earlier iteration $j < i$ even when $\boldsymbol{x}$ is not 2^j-sparse. But, $|J_{i+1}|$ being smaller only makes the algorithm run faster.

$\omega(\alpha, 1, \delta - \alpha) \leq \omega(\delta - 1, 1, 1)$ for any α, and so the claimed running time of $O(n^{\omega(\delta-1,1,1)+\varepsilon})$ follows since we only need to compute $O(\log n)$ such matrix products and the multiplicative $\log n$ factor gets absorbed by the ε in the exponent.

1.3.4 From Output-Sensitive MM to Fully Sparse MM

In Section 4, we extend our algorithms to the *fully sparse* case where we consider the sparsity of the $n \times n$ input matrices A, B in addition to the sparsity of AB. I.e., we are promised not only that $\|AB\|_0 \leq O(n^{\delta_{\text{out}}})$ for some $\delta_{\text{out}} \in [0, 2]$, but also that $\|A\|_0, \|B\|_0 \leq O(n^{\delta_{\text{in}}})$ for $\delta_{\text{in}} \in [0, 2]$.[8] For $\alpha, \beta, \gamma \in [0, 1]$, define $\omega_{\delta_{\text{in}}}(\alpha, \beta, \gamma)$ to be the infimum over $\omega' \geq 0$ such that it is possible to compute the product of an $n^\alpha \times n^\beta$ matrix A and an $n^\beta \times n^\gamma$ matrix B in $O(n^{\omega'})$ time when $\|A\|_0, \|B\|_0 \leq O(n^{\delta_{\text{in}}})$. We note that there are algorithms showing that $\omega_{\delta_{\text{in}}}(\alpha, \beta, \gamma)$ is smaller than $\omega(\alpha, \beta, \gamma)$ for sufficiently small values of δ_{in} [36, 22].

Theorem 18 (respectively, Theorem 19) extends our deterministic (respectively, randomized) algorithm in Theorem 1 (respectively, Theorem 2) to the fully sparse setting. Theorem 18 achieves a running time of roughly $O(n^{\omega_{\delta_{\text{in}}}(\delta_{\text{out}}/2,1,1)} + n^{\delta_{\text{out}}/2+1})$, which is equal to the $O(n^{\omega(\delta_{\text{out}}/2,1,1)})$ running time of Theorem 1 when $\delta_{\text{in}} = 2$, i.e., when there is no promise on the sparsity of A, B as in δ_{out}-OSMM.

Theorem 19 achieves a running time of roughly $O(n^{\delta_{\text{in}}} + n^\beta)$, for

$$\beta := \max_{\eta \in [0, \min\{\delta_{\text{out}}, 1\}]} \omega_{\delta_{\text{in}}}(\eta, 1, \delta_{\text{out}} - \eta) \ .$$

Although this expression looks intricate, it appears in [1, Lemma 3.1] (to which it is equivalent) and arises naturally in our FSMM proof of optimality; see Proposition 20. (For $\delta_{\text{out}} \geq 1$ and $\delta_{\text{in}} = 2$, we also get that $\omega_{\delta_{\text{in}}}(\eta, 1, \delta_{\text{out}} - \eta) \leq \omega(\delta_{\text{out}} - 1, 1, 1)$ by Corollary 8, and moreover $\omega(\delta_{\text{out}} - 1, 1, 1) \geq 2 \geq \delta_{\text{in}}$. So, we achieve the running time of Theorem 2.)

Extending our algorithms to the fully sparse setting essentially only uses basic facts about input-sparse matrix multiplication and properties of our compressed sensing scheme. Specifically, it uses the fact that our measurement matrix H has very sparse columns, and that on input $H\boldsymbol{x}$ for a t-sparse vector $\boldsymbol{x}$ of length n, $\mathcal{R}$ runs in time roughly t rather than time roughly n. In particular, we use that such a matrix H admits fast multiplication by any matrix A, and if A is sparse then HA is still nearly as sparse. (To achieve all of this, we use sparse representations of matrices and vectors.)

1.3.5 Instantiating our reductions

To instantiate our reductions and turn them into the algorithms summarized by Theorems 1 and 2, we need to give an explicit compressed sensing scheme and an algorithm for Column-Wise-MMV.

We use a compressed sensing scheme due to Berinde, Gilbert, Indyk, Karloff and Strauss [7], which requires certain expander graphs. To make the scheme of [7] deterministic and obtain good parameters, we instantiate it with the expander graphs later studied in Guruswami, Umans, and Vadhan [17]. In particular, our measurement matrices H correspond to the adjacency matrices of the bipartite expanders constructed in [17] from Parvaresh-Vardy codes. They are sparse and binary. The main theorem summarizing this scheme is Theorem 13.

[8] Without loss of generality we can take $\delta_{\text{out}} \leq 2\delta_{\text{in}}$.

For completeness, we present and analyze the expander construction in [17] and the sparse recovery algorithm in [7] in an appendix of the full version of this paper [6]. We show that they are both very efficient. Additionally, we observe that the sparse recovery algorithm in [7] works not just over the real numbers (as it was originally presented), but over arbitrary rings.

We emphasize the subtlety of identifying an appropriate compressed sensing scheme for our purposes. We need algorithms that compute the measurement matrix and perform sparse recovery not just in (arbitrary) polynomial time, but in nearly quadratic and nearly linear time, respectively. Moreover, we need these algorithms to be deterministic – many works on compressed sensing sample a random measurement matrix H (see, e.g., [9, 15]). Finally, for Theorem 2 we need multiplication by H to be very efficient.

The algorithm for Column-Wise-MMV that we use (stated formally in Lemma 16) is a modified version of Freivalds's algorithm for "plain" MMV [12]. This algorithm is the only use of randomness in our our randomized OSMM algorithm, and so if it could be derandomized efficiently then we would get a *deterministic* algorithm for δ-OSMM running in roughly $n^{\omega(\delta-1,1,1)}$ time. (Similarly, this algorithm is the only use of randomness in our randomized FSMM algorithm.)

1.3.6 An alternative randomized algorithm

We note that it is possible to implement a variant of our randomized algorithm that achieves a similar running time to Theorem 2 if it is possible to approximate the number of non-zero entries in each column of $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) = AB$ to within a constant factor efficiently.

In fact, we can compute estimates $\tilde{c}_i$ of the sparsity of each column $\boldsymbol{c}_i$ satisfying $1/2 \cdot \|\boldsymbol{c}_i\|_0 < \tilde{c}_i < 2 \cdot \|\boldsymbol{c}_i\|_0$ with high probability using a randomized algorithm from [31]. We can then use these estimates to multiply A and B efficiently. We first partition the columns of B (and therefore AB) into blocks B_j, where B_j consists of all columns $\boldsymbol{b}_i$ such that $2^{j-1} \leq \tilde{c}_i < 2^j$ for $j = 1, \ldots, r$, where $r \approx \log_2 n$. We then compute AB_j for each j using the algorithm for output-column-sparse matrix multiplication sketched earlier using the assumption (which holds with high probability) that each column of AB_j is 2^{j+1}-sparse.

We note that the column-sparsity-estimation-based algorithm sketched here uses randomness in a different way from Theorem 2; it uses randomness to estimate column sparsities and does not need to solve Column-Wise-MMV. However, we believe that the reduction from OSMM to Column-Wise-MMV that we use to prove Theorem 2 – which can be viewed as a search-to-decision reduction of sorts – is elegant and more amenable to derandomization. In particular, a strong resolution to the famous question of derandomizing Freivalds's algorithm would derandomize Theorem 2 as well.

2 Preliminaries

By $\mathbb{Z}^+$ we denote the set of positive integers. For $n \in \mathbb{Z}^+$, $[n]$ denotes the set $\{1, \ldots, n\}$. Let R be a ring, $\boldsymbol{v} \in R^n$ be a vector, and $M \in R^{m \times n}$ be a matrix. Then by $\|\boldsymbol{v}\|_0$ and $\|M\|_0$ we denote the number of non-zero entries in $\boldsymbol{v}$ and M, respectively. We say $\boldsymbol{v}$ (respectively, matrix M) is *t-sparse* if $\|\boldsymbol{v}\|_0 \leq t$ (respectively, if $\|M\|_0 \leq t$). We say that M is *t-column-sparse* if each of its columns is t-sparse. For an $n \in \mathbb{Z}^+$, $\mathbf{0}_n$ and $\mathbf{1}_n$ denote n-dimensional vectors with all zeros and all ones, respectively.

We use $\log(\cdot)$ to denote the logarithm base 2, i.e., $\log(2^n) = n$. We use the notation $\widetilde{O}(\cdot)$ to suppress polylogarithmic factors in the argument.

We state the running time bounds of our algorithms in terms of operations over a ring R, but these bounds immediately imply algorithms with similar running times (in the usual sense) over many standard domains. Over the integers with entries in $[-M, M]$ each operation requires at most $\mathrm{poly}(\log M, \log n)$ time, and over finite fields $\mathbb{F}_q$ each operations requires at most $\mathrm{poly}(\log q, \log n)$ time. Assigning unit cost to operations on real numbers is standard in the Real RAM model.

2.1 Matrix Multiplication

We first define the main two variants of matrix multiplication that we study.

▶ **Definition 3** (Output and Fully Sparse Matrix Multiplication). *Let R be a ring, and let $\delta_{in}, \delta_{out} \in [0, 2]$. We define the $(\delta_{in}, \delta_{out})$-*Fully Sparse Matrix Multiplication Problem *over R ($(\delta_{in}, \delta_{out})$-FSMM$_R$) as follows. The input consists of $A, B \in R^{n\times n}$ for some $n \in \mathbb{Z}^+$ with $\|A\|_0, \|B\|_0 \le O(n^{\delta_{in}})$ satisfying the promise that $\|AB\|_0 \le O(n^{\delta_{out}})$. The goal is to compute AB.*

*We additionally define the δ_{out}-*Output-Sparse Matrix Multiplication Problem *over R (δ_{out}-OSMM$_R$) as $(2, \delta_{out})$-FSMM$_R$.*

When both $\delta_{\mathrm{in}} = 2$ and $\delta_{\mathrm{out}} = 2$ this corresponds to the problem of matrix multiplication in its general form. Furthermore, note that since the product of two t-sparse matrices is t^2-sparse, we may without loss of generality assume that $\delta_{\mathrm{out}} \le 2\delta_{\mathrm{in}}$.

We next define rectangular matrix multiplication exponents, input-sparse rectangular matrix multiplication exponents, and the dual matrix multiplication exponent.

▶ **Definition 4** (Rectangular and Input-Sparse Rectangular Multiplication Exponents). *For constants $\alpha, \beta, \gamma \ge 0$, the* rectangular matrix multiplication exponent $\omega(\alpha, \beta, \gamma)$ *is the infimum over all $\omega' \ge 0$ such that the product of $A \in \mathbb{Z}^{n^\alpha \times n^\beta}$ and $B \in \mathbb{Z}^{n^\beta \times n^\gamma}$ can be computed using $O(n^{\omega'})$ arithmetic operations.*

We extend this notation and use $\omega_{\delta_{in}}(\alpha, \beta, \gamma)$ to denote the case when both A and B are promised to be $O(n^{\delta_{in}})$-sparse for $0 \le \delta_{in} \le \max\{\alpha + \beta, \beta + \gamma\}$.

We note that the *square matrix multiplication exponent* ω is simply $\omega := \omega(1, 1, 1)$, and that $\omega_{\delta_{\mathrm{in}}}(\alpha, \beta, \gamma) \le \omega_{\max\{\alpha+\beta, \beta+\gamma\}}(\alpha, \beta, \gamma) = \omega(\alpha, \beta, \gamma)$ for all $\alpha, \beta, \gamma \ge 0$.

▶ **Definition 5** (Dual Matrix Multiplication Exponent). *The* dual matrix multiplication exponent $\omega^\perp$ *is defined as*

$$\omega^\perp := \sup\{\omega' \ge 0 : \omega(1, 1, \omega') = 2\} .$$

A beautiful line of work [27, 3, 11, 26, 35, 2] has established strong bounds on multiplication exponents, in particular showing that

$$\omega < 2.371339 \quad [2] ,$$
$$\omega^\perp \ge 0.321334 \quad [35] .$$

2.2 Properties of Matrix Multiplication Exponents

Next we list several properties of rectangular matrix multiplication exponents $\omega(\cdot, \cdot, \cdot)$.

▶ **Proposition 6** ([29]). *Let $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \lambda \ge 0$. The following hold:*

1. $\max(\alpha_1 + \alpha_2, \alpha_1 + \alpha_3, \alpha_2 + \alpha_3) \le \omega(\alpha_1, \alpha_2, \alpha_3) \le \alpha_1 + \alpha_2 + \alpha_3$.
2. $\omega(\lambda\alpha_1, \lambda\alpha_2, \lambda\alpha_3) = \lambda \cdot \omega(\alpha_1, \alpha_2, \alpha_3)$.

3. $\omega(\alpha_1+\beta_1,\alpha_2+\beta_2,\alpha_3+\beta_3) \le \omega(\alpha_1,\alpha_2,\alpha_3)+\omega(\beta_1,\beta_2,\beta_3)$.
4. *For any permutation* $\pi:[3]\to[3]$, $\omega(\alpha_1,\alpha_2,\alpha_3)=\omega(\alpha_{\pi(1)},\alpha_{\pi(2)},\alpha_{\pi(3)})$.
5. $\omega(\alpha,\beta,\gamma)$ *is continuous and non-decreasing in its arguments for* $\alpha,\beta,\gamma \ge 0$.

We will make use of the following upper bound on $\omega(\alpha,1,1)$.

▶ **Theorem 7** ([29, 25]). *Let* $\alpha\in[0,1]$. *Then*

$$\omega(\alpha,1,1) \le \begin{cases} 2 & \text{if } 0\le\alpha\le\omega^\perp\ , \\ 2+\dfrac{\omega-2}{1-\omega^\perp}\cdot(\alpha-\omega^\perp) \le 1.825+0.548\alpha & \text{if } \omega^\perp<\alpha\le 1\ . \end{cases}$$

We next give an upper bound on rectangular matrix multiplication exponents $\omega(1,\alpha,\beta)$ with three potentially distinct arguments.

▶ **Corollary 8.** *Let* $\alpha,\beta,\gamma\in[0,1]$ *be such that* $\beta+\gamma\ge 1$. *Then* $\omega(\alpha,\beta,\gamma)\le\omega(\alpha,1,\beta+\gamma-1)$.

Proof. If $\beta=\gamma=1$, the statement holds trivially. Otherwise we can assume that $\beta+\gamma<2$. Let $\lambda=(1-\gamma)/(2-\beta-\gamma)\ge 0$. Then we have that

$$\begin{aligned}\omega(\alpha,\beta,\gamma) &= \omega(\lambda\alpha+(1-\lambda)\alpha,\lambda+(1-\lambda)(\beta+\gamma-1),\lambda(\beta+\gamma-1)+(1-\lambda)) \\ &\le \omega(\lambda\alpha,\lambda,\lambda(\beta+\gamma-1))+\omega((1-\lambda)\alpha,(1-\lambda)(\beta+\gamma-1),(1-\lambda)) \\ &= \lambda\cdot\omega(\alpha,1,\beta+\gamma-1)+(1-\lambda)\cdot\omega(\alpha,\beta+\gamma-1,1) \\ &= \omega(\alpha,1,\beta+\gamma-1)\ ,\end{aligned}$$

where the inequality uses Proposition 6, Item 3, the following equality uses Proposition 6, Item 2, and the last one uses Proposition 6, Item 4. ◀

2.3 Input-Sparse Matrix Multiplication

In this paper, we assume *sparse representations* of vectors and matrices. Specifically, we represent a vector $\boldsymbol{x}\in R^n$ by a list of index-value pairs of its non-zero entries. We represent a matrix $A=(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_n)\in R^{m\times n}$ using lists of the non-zero entries of each column $\boldsymbol{a}_i$, sorted by i.

We next give a basic algorithm for matrix multiplication when one of the matrices is column-sparse. This algorithm is folklore, and appears in work at least as early as Gustavson [18].

▶ **Proposition 9** (Naïve Input-Sparse Matrix Multiplication). *Let* $A\in R^{m\times n}, B\in R^{n\times p}$ *for a ring* R *and* $m,n,p\in\mathbb{Z}^+$. *Let* A *be* t*-column-sparse for* $t\in\mathbb{Z}^+$. *Then* $\|AB\|_0\le t\cdot\|B\|_0$, *and there is a deterministic algorithm that computes* AB *using* $t\cdot\|B\|_0\cdot\mathrm{poly}(\log n)$ *arithmetic operations over* R.

Proof. For $i\in[n]$, let b_i be the number of non-zero elements in the ith row of B. If we consider the multiplication of A and B as a sum of outer products, then for every $k\in[n]$, the outer product of the kth column of A and the kth row of B can be computed using at most $t\cdot b_k$ multiplications, and has sparsity at most $t\cdot b_k$. Therefore, $\|AB\|_0\le\sum_{i\in[n]}t\cdot b_k=t\cdot\|B\|_0$. The running time of this algorithm is bounded by $t\cdot\|B\|_0\cdot\mathrm{poly}(\log n)$, where the $\mathrm{poly}(\log n)$ factor accounts for the sparse representation. ◀

Next, we present known bounds on the complexity of matrix multiplication when the input matrices are sparse. The following bound was proven for square matrices in [36], and was generalized to rectangular matrices in [22] (see also [1, Lemma 4.2]).

▶ **Theorem 10** ([36, 22]). *Let $\alpha, \beta, \gamma > 0$ and $0 \le \delta_{in} \le \max\{\alpha+\beta, \beta+\gamma\}$. Then*

$$\omega_{\delta_{in}}(\alpha,\beta,\gamma) \le \min_{\eta\in[0,\beta]} \max\{\omega(\alpha,\eta,\gamma),\, 2\delta_{in}-\eta\}\ .$$

Next, we list two additional bounds on sparse rectangular matrix multiplication exponents $\omega_{\delta_{\text{in}}}(\cdot,\cdot,\cdot)$.

▶ **Proposition 11.** *Let $\alpha, \beta, \gamma > 0$. The following hold:*

1. *If $0 \le \delta_{in} \le \max\{\alpha+\beta, \beta+\gamma\}$, then $\omega_{\delta_{in}}(\alpha,\beta,\gamma) \le \delta_{in} + \min\{\alpha,\gamma\}$.*
2. *If $\delta_{in} \ge \max\{\alpha,\gamma\}$, then $\omega_{\delta_{in}}(\alpha,\beta,\gamma) \ge \alpha+\gamma$.*

Proof. The first bound follows from Proposition 9 by noting that every $n^\alpha \times n^\beta$ matrix is n^α-column-sparse (and that the transpose of an $n^\beta \times n^\gamma$ matrix is n^γ-column-sparse).

For the second bound, consider $A = (\mathbf{1}, \mathbf{0}, \ldots, \mathbf{0}) \in R^{n^\alpha \times n^\beta}$ and $B = (\mathbf{1}, \mathbf{0}, \ldots, \mathbf{0})^T \in R^{n^\beta \times n^\gamma}$. Note that $\|A\|_0, \|B\|_0 \le O(n^{\delta_{\text{in}}})$, and $\|AB\|_0 = n^{\alpha+\gamma}$. Therefore, any algorithm computing AB must have running time at least $\Omega(n^{\alpha+\gamma})$. ◀

3 Output-Sparse Matrix Multiplication

We now present our main technical results on OSMM. In Section 3.1, we formally define compressed sensing schemes and state our main theorem about a concrete such scheme. In Section 3.2, we give a reduction from δ-OSMM to compressed sensing and instantiate the reduction to give our roughly $n^{\omega(\delta/2,1,1)}$-time deterministic algorithm for δ-OSMM, proving Theorem 1. In Section 3.3, we introduce a variant of matrix multiplication verification (MMV) and give a reduction from OSMM to compressed sensing and this MMV variant. We then instantiate the reduction to give our roughly $n^{\omega(\delta-1,1,1)}$-time randomized algorithm for δ-OSMM, proving Theorem 2.

3.1 Compressed Sensing

We will use the following formalization of a compressed sensing scheme.

▶ **Definition 12.** *Let R be a ring and let $m = m(n,t)$ be a non-decreasing, integer-valued function. An m-*compressed sensing scheme *over R is a pair of algorithms $(\mathcal{M}, \mathcal{R})$ that work as follows. On inputs n and t in unary with $1 \le t \le n$, $\mathcal{M}$ outputs a matrix $H \in R^{m\times n}$. On input $H\boldsymbol{x}$ for a vector $\boldsymbol{x} \in R^n$ satisfying $\|\boldsymbol{x}\|_0 \le t$, $\mathcal{R}$ outputs $\boldsymbol{x}$.*[9]

Here the matrix H output by $\mathcal{M}$ is called a *measurement matrix*, and $\mathcal{R}$ is called a *sparse recovery algorithm*. We will crucially use the following theorem about an explicit compressed sensing scheme from [7] (which we instantiate with a construction of expanders from [17]) in both of our main algorithms. The claim in Theorem 13 follows implicitly by combining [7, 17], but we include a full proof for completeness. However, the proof is somewhat involved and so we defer it to an appendix of the full version of this paper [6].

[9] In fact, the sparse recovery algorithm $\mathcal{R}$ needs to know the value of n, and often it also needs t and H as well. This is usually handled by having the algorithm $\mathcal{M}$ also compute a string σ as preprocessing that is passed as auxiliary input to $\mathcal{R}$. Without loss of generality, we assume that σ encodes H, n, and t, but it could also encode more. So, formally we should write $\mathcal{R}(H\boldsymbol{x}, \sigma)$ for running $\mathcal{R}$ on $H\boldsymbol{x}$, but for conciseness we often abuse notation and simply write $\mathcal{R}(H\boldsymbol{x})$.

▶ **Theorem 13** ([7]). *Let R be a ring and let $\alpha > 0$ be a constant. There is a deterministic m-compressed sensing scheme $(\mathcal{M}, \mathcal{R})$ over R with $m = m(n,t) = t^{1+\alpha} \cdot \mathrm{poly}(\log n)$. The algorithm $\mathcal{M}$ runs in time $\widetilde{O}(n)$, and the algorithm $\mathcal{R}$ performs $t^{1+\alpha} \cdot \mathrm{poly}(\log n)$ arithmetic operations over R.*

Furthermore, the measurement matrix $H := \mathcal{M}(1^n, 1^t)$ is binary and d-column-sparse for $d = \mathrm{poly}(\log n)$, and therefore can be multiplied by an arbitrary vector $\boldsymbol{x} \in R^n$ using $\|\boldsymbol{x}\|_0 \cdot \mathrm{poly}(\log n)$ addition operations over R.[10]

3.2 Deterministic OSMM via a Reduction to Compressed Sensing

In this section, we start by giving our two-pass reduction from OSMM to compressed sensing in Algorithm 1. One may also view this reduction as a meta-algorithm. Making it into an explicit algorithm requires using an explicit compressed sensing scheme (we note this in the "Subroutine" heading in the specification of Algorithm 1).

Algorithm 1 Deterministic output-sparse matrix multiplication over a ring R.

Input: Matrices $A, B \in R^{n \times n}$ over a ring R, and $t \in \mathbb{Z}^+$ such that $\|AB\|_0 \le t^2$.
Subroutine: An m-compressed sensing scheme $(\mathcal{M}, \mathcal{R})$ over R.
Output: The product $C = AB$.

1 Compute a measurement matrix $H \in R^{m \times n} := \mathcal{M}(1^n, 1^t)$.
2 Compute $D = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n) := HAB$ using fast rectangular MM.
3 Compute $D' := (\mathcal{R}(\boldsymbol{d}_1), \ldots, \mathcal{R}(\boldsymbol{d}_n))$. // t-sparse columns of $C = AB$ agree with D'.
4 Compute $F = (\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n) := (H(AB - D')^T)$ using fast rectangular MM.
5 Compute $F' := (\mathcal{R}(\boldsymbol{f}_1), \ldots, \mathcal{R}(\boldsymbol{f}_n))$. // $F' = (AB - D')^T$.
6 **return** $D' + (F')^T$.

We next analyze Algorithm 1.

▶ **Theorem 14.** *Let R be a ring, and let $A, B \in R^{n \times n}$ be matrices satisfying $\|AB\|_0 \le t^2$ for $t \in \mathbb{Z}^+$ with $t^2 \le O(n^\delta)$ for some $\delta \in [0,2]$. Let $(\mathcal{M}, \mathcal{R})$ be an m-compressed sensing scheme over R with $m = m(n,t) \le O(t \cdot n^\alpha)$ for some $\alpha > 0$. Then on input A, B, Algorithm 1 outputs the matrix product $C := AB$. Furthermore, if $\mathcal{M}$ runs in time $T_{\mathcal{M}}(n,t)$ and $\mathcal{R}$ runs in time $T_{\mathcal{R}}(n,t)$ then for any constant $\varepsilon > 0$, Algorithm 1 performs*

$$O(T_{\mathcal{M}}(n,t) + n \cdot T_{\mathcal{R}}(n,t) + n^{\omega(\delta/2+\alpha,1,1)+\varepsilon})$$

arithmetic operations over R.

Proof. We start by proving correctness. Let $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$, $D' = (\boldsymbol{d}'_1, \ldots, \boldsymbol{d}'_n)$, and $F' = (\boldsymbol{f}'_1, \ldots, \boldsymbol{f}'_n)$. We assume without loss of generality that $\mathcal{R}$ outputs *some* vector in R^n on every input $\boldsymbol{y} \in R^m$ (including when $\boldsymbol{y} \neq H\boldsymbol{x}$ for any t-sparse $\boldsymbol{x} \in R^n$).

We note that C has at most t columns $\boldsymbol{c}_i$ that are not t-sparse, since otherwise we would have $\|C\|_0 > t^2$. Furthermore, by the specification of $\mathcal{R}$, $\boldsymbol{d}'_i = \mathcal{R}(H\boldsymbol{c}_i) = \boldsymbol{c}_i$ for each $i \in [n]$ such that $\boldsymbol{c}_i$ is t-sparse. It follows that every row of $C - D'$ is t-sparse, and therefore that every column of $(AB - D')^T$ is t-sparse. From this and again using the specification of $\mathcal{R}$, we get that $F' = (AB - D')^T$. Correctness then follows by noting that $C = AB = D' + (F')^T$.

[10] Assuming that H and $\boldsymbol{x}$ are given in sparse representation by lists of non-zero entries (as in the output of the algorithm $\mathcal{M}$).

We next analyze the time complexity of Algorithm 1. It makes one call to $\mathcal{M}$ on input $(1^n, 1^t)$, which takes time at most $T_{\mathcal{M}}(n,t)$, and $2n$ calls to $\mathcal{R}$, which takes time at most $2n \cdot T_{\mathcal{R}}(n,t)$. Furthermore, because $H \in R^{m\times n}$ for $t = O(n^{\delta/2})$ and $m \le O(t \cdot n^{\alpha}) = O(n^{\delta/2+\alpha})$, HAB and $H(AB - D')^T$ can be computed in time $O(n^{\omega(\delta/2+\alpha,1,1)+\varepsilon})$ for any constant $\varepsilon > 0$ using fast rectangular matrix multiplication. All other operations run in $O(n^2) \le O(n^{\omega(\delta/2+\alpha,1,1)})$ time. The result follows by summing the running time bounds for each of these components. ◀

Our main theorem about solving OSMM deterministically follows as a corollary.

▶ **Theorem 1** (Deterministic OSMM). *Let $\delta \in [0,2]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a deterministic algorithm for solving δ-OSMM$_R$ on $n \times n$ matrices that performs $O(n^{\omega(\delta/2,1,1)+\varepsilon})$ operations over R.*

Proof. We instantiate the compressed sensing scheme subroutine needed in Algorithm 1 with the scheme described in Theorem 13. Let $\alpha > 0$ be a constant, which we will set later, and let α' be a constant satisfying $0 < \alpha' < \alpha$. From Theorem 13, we have that there exists a deterministic m-compressed sensing scheme where $m = m(n,t) = t^{1+\alpha'} \cdot \mathrm{poly}(\log n) \le O(tn^{\alpha})$, with $T_{\mathcal{M}}(n,t) \le T_{\mathcal{M}}(n,n) \le \widetilde{O}(n) \le O(n^{1+\alpha})$ and $T_{\mathcal{R}}(n,t) \le T_{\mathcal{R}}(n,n) \le O(n^{1+\alpha})$. Furthermore, we have that $\omega(\delta/2+\alpha,1,1) \le \omega(\delta/2,1,1) + \omega(\alpha,0,0) = \omega(\delta/2,1,1) + \alpha$ by Proposition 6, Items 1 and 3. Using the fact that $\omega(\delta/2,1,1) \ge 2$ for all $\delta \in [0,2]$, we then have that for any constant $\varepsilon' > 0$ the algorithm performs at most

$$\begin{aligned} O(T_{\mathcal{M}}(n,t) + n \cdot T_{\mathcal{R}}(n,t) + n^{\omega(\delta/2+\alpha,1,1)+\varepsilon'}) &\le O(n^{1+\alpha} + n^{2+\alpha} + n^{\omega(\delta/2,1,1)+\alpha+\varepsilon'}) \\ &\le O(n^{\omega(\delta/2,1,1)+\alpha+\varepsilon'}) \end{aligned}$$

operations over R. To prove the claim, we need to show that the algorithm performs at most $O(n^{\omega(\delta/2,1,1)+\varepsilon})$ operations. This follows by choosing $\varepsilon', \alpha > 0$ such that $\varepsilon' + \alpha \le \varepsilon$. ◀

3.3 Randomized OSMM via a Reduction to a Variant of MMV and Compressed Sensing

In this section, we get a roughly $n^{\omega(\delta-1,1,1)}$-time algorithm for δ-OSMM by reducing to a variant of MMV and to compressed sensing.

3.3.1 Column-Wise Matrix Multiplication Verification

In this section, we introduce and give an algorithm for the variant of MMV in which the goal is to check whether each column of AB is equal to its corresponding column in C'.

▶ **Definition 15.** *For a ring R, the Column-Wise MMV problem over R (Column-Wise-MMV$_R$) is defined as follows. The input consists of matrices $A \in R^{m\times n}$, $B \in R^{n\times p}$, and $C' = (\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_p) \in R^{m\times p}$. Let $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_p) := AB$. The goal is to output the set $J \subseteq [p]$ of indices of columns on which C and C' differ, i.e.,*

$$J := \{j \in [p] : \boldsymbol{c}_j \neq \boldsymbol{c}'_j\} \ .$$

We note that "normal" MMV corresponds to deciding whether $J = \emptyset$ in Definition 15, and so MMV easily reduces to Column-Wise-MMV. We next argue that a slight variant of Freivalds's algorithm [12] also solves Column-Wise-MMV in nearly quadratic time.

▶ **Lemma 16** (Freivalds's algorithm for Column-Wise-MMV)**.** *Let R be a ring and let $c \in \mathbb{Z}^+$ be a constant. For any $m, n, p \in \mathbb{Z}^+$, there is a randomized algorithm that solves* Column-Wise-MMV$_R$ *on matrices $A \in R^{m\times n}, B \in R^{n\times p}, C' \in R^{m\times p}$ performing $O(\max\{mn, np\} \cdot \log(mnp))$ arithmetic operations over R and has a success probability of at least $1 - 1/(mnp)^c$. In particular, when $m, p = O(n)$, the algorithm performs $O(n^2 \log n)$ arithmetic operations over R and has a success probability of at least $1 - 1/n^c$.*

Proof. The algorithm works as follows. First, it samples a uniformly random matrix $X \sim \{0,1\}^{N\times m}$ for $N := \lceil (c+1)\log(mnp)\rceil$. It then computes $D = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_p) := X(AB - C')$, and outputs $J' := \{i \in [p] : \boldsymbol{d}_i \neq \boldsymbol{0}\}$.

We start by showing correctness of the algorithm, i.e., that $J' = J$ for J as defined in Definition 15. For a fixed non-zero vector $\boldsymbol{y} \in R^m$ and a uniformly random $\boldsymbol{x} \sim \{0,1\}^m$, $\Pr[\langle \boldsymbol{x}, \boldsymbol{y}\rangle \neq 0] \geq 1/2$. Indeed, this follows by noting that each term $x_i y_i$ in $\langle \boldsymbol{x}, \boldsymbol{y}\rangle = \sum_{i=1}^m x_i y_i$ for non-zero y_i is equal to 0 with probability $1/2$ and an induction argument. So, for any non-zero column $\boldsymbol{y}_i$ of $AB - C'$, we get that $\Pr[X\boldsymbol{y}_i = \boldsymbol{0}] \leq 1/2^N$. (If $\boldsymbol{y}_i = 0$, then $X\boldsymbol{y}_i = \boldsymbol{0}$ with probability 1.) Taking a union bound over the at most p non-zero columns of $AB - C'$, we then get that the probability that there exists a non-zero column $\boldsymbol{y}_i$ of $AB - C'$ such that $\boldsymbol{d}_i = \boldsymbol{0}$ is at most $p/2^N \leq p/(mnp)^{c+1} \leq 1/(mnp)^c$, as needed.

Furthermore, the algorithm runs in $O(N \cdot \max\{mn, np\}) = O(\max\{mn, np\} \cdot \log(mnp))$ time, which finishes the analysis. ◀

3.3.2 The Reduction and its Analysis

We next give our reduction from OSMM (and in fact, simply MM) to Column-Wise-MMV and compressed sensing in Algorithm 2. We sometimes use sets to index the rows and columns of a matrix. I.e., for finite sets I and J we use notation like $A \in R^{I\times J}$ to denote a matrix over a ring R with $|I|$ rows indexed by I and $|J|$ columns indexed by J. Similarly, we use notation like $A_{I',J'}$ to denote the submatrix of $A \in R^{I\times J}$ with rows indexed by $I' \subseteq I$ and columns indexed by $J' \subseteq J$.

In Theorem 17, we analyze Algorithm 2. We again emphasize that its correctness does not depend on the sparsity of AB; only its running time does. Furthermore, the same algorithm works regardless of the sparsity of AB.

▶ **Theorem 17.** *Let $A, B \in R^{n\times n}$ over a ring R, let $(\mathcal{M}, \mathcal{R})$ be an m-compressed sensing scheme with $m = m(n,t) \leq O(tn^\alpha)$ for some $\alpha \in (0, \omega^\perp)$, and let* ColMMV *be an algorithm for* Column-Wise-MMV *over R. Then on input $A, B \in R^{n\times n}$, Algorithm 2 outputs the matrix product $C = AB$.*

Furthermore, let $T_{\mathcal{M}}(n,t)$ and $T_{\mathcal{R}}(n,t)$ denote the running times of $\mathcal{M}$ and $\mathcal{R}$, respectively, let $T_H(n,t)$ denote the time needed to multiply $H := \mathcal{M}(1^n, 1^t)$ by an $n \times n$ matrix, and let $T_V(m,n,p)$ denote the running time of ColMMV *on input matrices $A \in R^{m\times n}, B \in R^{n\times p}, C \in R^{m\times p}$. If $\|AB\|_0 \leq O(n^\delta)$ for $0 \leq \delta \leq 2$ then the algorithm performs*

$$\widetilde{O}(n^{\beta+\varepsilon} + T_{\mathcal{M}}(n,n) + n \cdot T_{\mathcal{R}}(n,n) + T_H(n,n) + T_V(n,n,n))$$

arithmetic operations over R for any constant $\varepsilon > 0$, where

$$\beta := \begin{cases} 2 & \text{if } \delta + \alpha \leq 1 + \omega^\perp \text{ ,} \\ \omega(\delta + \alpha - 1, 1, 1) & \text{otherwise .} \end{cases}$$

Proof. Assume without loss of generality that $\delta \geq 1$. If not, set $\delta = 1$, and using that $\alpha \leq \omega^\perp$ conclude that $\beta = 2$. Let $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n)$.

Algorithm 2 Matrix multiplication using column-wise matrix multiplication verification.

Input: Matrices $A, B \in R^{n\times n}$ over a ring R.
Subroutine: An m-compressed sensing scheme $(\mathcal{M}, \mathcal{R})$ over R.
Subroutine: An algorithm ColMMV for Column-Wise-MMV.
Output: The product AB.

Set $J_0 := [n]$.
Initialize a matrix $C' = (\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_n) \in R^{n\times n}$ arbitrarily.
for $i = 0, 1, \ldots, \lceil \log n \rceil$ **do**
  Set $t := 2^i$.
  Compute $H := \mathcal{M}(1^n, 1^t) \in R^{m\times n}$. // $m = m(n,t)$.
  Compute $B' := B_{[n],J_i} \in R^{[n]\times J_i}$. // Restrict B to columns indexed by J_i.
  Compute $D = (\boldsymbol{d}_j)_{j\in J_i} := HAB' \in R^{[m]\times J_i}$.
  Compute $F = (\boldsymbol{f}_j)_{j\in J_i} := (\mathcal{R}(\boldsymbol{d}_j))_{j\in J_i} \in R^{[n]\times J_i}$.
  Compute $J_{i+1} := \text{ColMMV}(A, B', F)$. // Indices of columns that differ in AB', F.
  for $j \in J_i \setminus J_{i+1}$ **do**
    Set $\boldsymbol{c}'_j := \boldsymbol{f}_j$.
return C'

We argue that after the ith iteration of the for loop, $\boldsymbol{c}'_j = \boldsymbol{c}_j$ for all $j \in [n]$ such that $\|\boldsymbol{c}_j\|_0 \le 2^i$. Indeed, this follows from the fact that for such an index j, $\boldsymbol{f}_j = \mathcal{R}(H\boldsymbol{c}_j) = \boldsymbol{c}_j$ by the definition of a compressed sensing scheme with $t = 2^i$, and therefore we have that $j \notin J_{i+1}$. (As in the proof of Theorem 14 we assume that $\mathcal{R}$ always outputs *some* vector in R^n, regardless of its input.) In particular, because $\|\boldsymbol{c}_j\|_0 \le n \le 2^{\lceil \log n \rceil}$ for all columns $\boldsymbol{c}_j$, $\boldsymbol{c}'_j = \boldsymbol{c}_j$ for all $j \in [n]$ after the last iteration of the for loop, which implies that $C' = C$, as needed.

We next turn to analyzing the algorithm's running time. We upper bound the worst-case running time of a single iteration (iteration i) of the for loop, and note that the loop performs $O(\log n)$ iterations. The $\mathcal{M}(1^n, 1^t)$ call takes time $T_{\mathcal{M}}(n,t) \le T_{\mathcal{M}}(n,n)$, the $|J_i| \le n$ calls to $\mathcal{R}$ take time at most $|J_i| \cdot T_{\mathcal{R}}(n,t) \le n \cdot T_{\mathcal{R}}(n,n)$, and the call to ColMMV takes time at most $T_V(n, n, |J_i|) \le T_V(n,n,n)$.

It remains to upper bound the complexity of computing HAB' in the ith iteration, which we compute parenthesized as $(HA)B'$. Computing $A' := HA$ takes $T_H(n,t) \le T_H(n,n)$ time by assumption,[11] and it remains to compute $A'B'$, which we do using rectangular matrix multiplication. For $i = 0$, we have $t = 1$ and $m = m(n,t) = O(n^\alpha)$. Therefore, computing $A'B'$ takes $O(n^{\omega(\alpha,1,1)+\varepsilon}) = O(n^{\omega(\delta+\alpha-1,1,1)+\varepsilon})$ time. For $i \ge 1$, we have $t = 2^i$ and $(t/2) \cdot |J_i| \le \|AB\|_0$, which holds because J_i consists of indices of columns of AB with more than $t/2$ non-zero entries.

Let $\eta, \gamma \ge 0$ be such that $t = O(n^\eta)$ and $|J_i| = O(n^\gamma)$. Then because $A' \in R^{m\times n}$ for $m = m(n,t) = O(tn^\alpha) = O(n^{\eta+\alpha})$ and $B' \in R^{n\times|J_i|}$, $A'B'$ can be computed in $O(n^{\omega(\eta+\alpha,1,\gamma)+\varepsilon})$ time for any constant $\varepsilon > 0$. Furthermore, since $(t/2) \cdot |J_i| \le \|AB\|_0 \le O(n^\delta)$, we have that $\eta + \gamma \le \delta$. If $\eta + \alpha + \gamma \le 1$, then by Proposition 6, Item 1, $\omega(\eta+\alpha, 1, \gamma) \le \eta + \alpha + 1 + \gamma \le 2$. Otherwise, if $\eta + \alpha + \gamma > 1$, then by Proposition 6, Item 4 and Corollary 8, $\omega(\eta+\alpha, 1, \gamma) \le \omega(\eta+\alpha+\gamma-1, 1, 1) \le \omega(\delta+\alpha-1, 1, 1)$, as needed. ◀

[11] When instantiating this reduction each matrix H will be sparse. Such matrices admit fast multiplication.

Our main theorem about solving OSMM via a randomized algorithm follows as a corollary.

▶ **Theorem 2** (Randomized OSMM). *Let $\delta \in [0,2]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a randomized algorithm for solving δ-OSMM$_R$ on $n \times n$ matrices that performs $O(n^{\beta+\varepsilon})$ operations over R, where*

$$\beta := \begin{cases} 2 & \text{if } \delta \leq 1 + \omega^{\perp} \ , \\ \omega(\delta - 1, 1, 1) & \text{otherwise} \ . \end{cases}$$

Proof. Assume without loss of generality that $\delta \geq 1$. If not, set $\delta = 1$. We instantiate the compressed sensing scheme subroutine needed in Algorithm 2 with the scheme described in Theorem 13. The running time analysis follows from a similar argument as in the proof of Theorem 1, combined with the guarantee that $T_H(n,n) \leq \widetilde{O}(n^2)$ from Theorem 13 and the guarantee that $T_V(n,n,n) \leq \widetilde{O}(n^2)$ from Lemma 16. In particular, the fact that H can be multiplied by an arbitrary vector $\boldsymbol{x}$ in $\widetilde{O}(\|\boldsymbol{x}\|_0)$ time implies that H can be multiplied by an arbitrary matrix $A \in R^{n \times n}$ in $\widetilde{O}(n^2)$ time. This implies that $T_H(n,n) \leq \widetilde{O}(n^2)$.

Furthermore, the algorithm in Lemma 16 is randomized and succeeds with probability $1 - 1/n^c$ for any constant $c > 0$. Fix any such $c > 0$. Algorithm 2 makes $O(\log n)$ calls to ColMMV, and so by taking a union bound we get that all of these calls succeed with probability at least $1 - O(\log n / n^c) = 1 - o(1)$. It follows that Algorithm 2 succeeds with probability $1 - o(1)$, as needed. ◀

4 Fully Sparse Matrix Multiplication

In this section, we describe how to extend our main algorithms for OSMM to the fully sparse setting when the input matrices A, B are promised not only to satisfy $\|AB\|_0 \leq O(n^{\delta_{\text{out}}})$ but also $\|A\|_0, \|B\|_0 \leq O(n^{\delta_{\text{in}}})$ for some $\delta_{\text{in}} \in [0,2]$.

4.1 Deterministic FSMM

We first give our deterministic algorithm for FSMM, which follows by instantiating Algorithm 1 with the compressed sensing scheme in Theorem 13 as in the proof of Theorem 1.

▶ **Theorem 18** (Deterministic FSMM). *Let $\delta_{in}, \delta_{out} \in [0,2]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a deterministic algorithm for solving $(\delta_{in}, \delta_{out})$-FSMM$_R$ on $n \times n$ matrices that performs $O(n^{\beta+\varepsilon})$ operations over R, where*

$$\beta := \max\{\delta_{out}/2 + 1, \omega_{\delta_{in}}(\delta_{out}/2, 1, 1)\} \ .$$

Proof sketch. We refer to Algorithm 1, instantiated with the compressed sensing scheme in Theorem 13, and show how to implement its operations under the assumption that $\|A\|_0, \|B\|_0 \leq O(n^{\delta_{\text{in}}})$. Let $\alpha > 0$ be a small constant to be set in the analysis. We first note that $T_{\mathcal{M}}(n,t) \leq \widetilde{O}(n)$ as before.

We now analyze the time needed to compute $D = HAB$. By Theorem 13, H is poly$(\log n)$-column-sparse, and so it is possible to compute $A' := HA$ in $\widetilde{O}(n^{\delta_{\text{in}}})$ time by Proposition 9. Since $\|A'\|_0 = \widetilde{O}(n^{\delta_{\text{in}}})$ and $H, A' \in R^{m \times n}$ for $m = O(n^{\delta_{\text{out}}/2+\alpha})$, it is possible to compute the product $A'B$ in $O(n^{\omega_{\delta_{\text{in}}}(\delta_{\text{out}}/2,1,1)+\varepsilon})$ time for any constant $\varepsilon > 0$ (assuming that $\alpha > 0$ is chosen to be sufficiently small; see the proof of Theorem 1). We note that $\omega_{\delta_{\text{in}}}(\delta_{\text{out}}/2, 1, 1) \geq \delta_{\text{in}}$ (since simply reading in an $n \times n$ matrix B with $\Theta(n^{\delta_{\text{in}}})$ non-zero entries takes $\Omega(n^{\delta_{\text{in}}})$ time), and so computing HAB takes $O(n^{\omega_{\delta_{\text{in}}}(\delta_{\text{out}}/2,1,1)+\varepsilon})$ time overall.

Furthermore, by Theorem 13, $T_{\mathcal{R}}(t,n) \leq t^{1+\alpha} \cdot \text{poly}(\log n) \leq \widetilde{O}(n^{\delta_{\text{out}}/2+\alpha})$ since $t = O(n^{\delta_{\text{out}}/2})$, and so computing D' and F' takes $\widetilde{O}(n^{\delta_{\text{out}}/2+\alpha+1})$ time. Finally, we analyze the time needed to compute F. Computing $H(AB)^T$ can be done in $O(n^{\omega_{\delta_{\text{in}}}(\delta_{\text{out}}/2,1,1)+\varepsilon})$ time for any constant $\varepsilon > 0$ as with computing D. Furthermore, D' has at most $O(n^{\delta_{\text{out}}/2})$ non-zero columns, and so, again using the fact that H is $\text{poly}(\log n)$-column-sparse, it takes $\widetilde{O}(n^{\delta_{\text{out}}/2+1})$ time to compute $H(D')^T$. The claim follows. ◀

We recover Theorem 1 from Theorem 18 by noting that $\omega_2(\delta_{\text{out}}/2,1,1) = \omega(\delta_{\text{out}}/2,1,1)$ and that $\omega(\delta_{\text{out}}/2,1,1) \geq \delta_{\text{out}}/2+1$ (by Proposition 6, Item 1).

Using the bounds on $\omega_{\delta_{\text{in}}}(\delta_{\text{out}}/2,1,1)$ from Proposition 11, Item 1 and Theorem 10, we conclude that for every $\varepsilon > 0$, $(\delta_{\text{in}}, \delta_{\text{out}})$-FSMM$_R$ can be solved using $O(n^{\delta_{\text{out}}/2+1+\varepsilon} + n^{\delta_{\text{out}}/2+\delta_{\text{in}}+\varepsilon})$ or $O(n^{\delta_{\text{out}}/2+1+\varepsilon} + n^{\sigma+\varepsilon})$ operations over R, where

$$\sigma \leq \min_{\eta \in [0,1]} \max\{\omega(\delta_{\text{out}}/2, \eta, 1),\ 2\delta_{\text{in}} - \eta\}\ . \tag{2}$$

4.2 Randomized FSMM

We now give our randomized algorithm for FSMM as Theorem 19. It follows by instantiating Algorithm 2 with the compressed scheme in Theorem 13 as in the proof of Theorem 2. We recall that Theorem 19 is equivalent to [1, Lemma 3.1], although the algorithm that it is based on uses different techniques.

▶ **Theorem 19** (Randomized FSMM). *Let $\delta_{in} \in [1,2], \delta_{out} \in [0, 2\delta_{in}]$, let R be a ring, and let $\varepsilon > 0$ be a constant. There is a randomized algorithm for solving $(\delta_{in}, \delta_{out})$-FSMM$_R$ on $n \times n$ matrices that performs $O(n^{\delta_{in}+\varepsilon} + n^{\beta+\varepsilon})$ operations over R, where*

$$\beta := \max_{\substack{\eta_1,\eta_2 \in [0,1]\,, \\ \eta_1+\eta_2=\delta_{out}}} \omega_{\delta_{in}}(\eta_1, 1, \eta_2)\ .$$

Proof sketch. We refer to Algorithm 2, instantiated with the compressed sensing scheme in Theorem 13, and show how to implement its operations under the assumption that $\|A\|_0, \|B\|_0 \leq O(n^{\delta_{\text{in}}})$.

As in the proof of Theorem 2, we upper bound the worst-case running time of a single iteration (iteration i) of the for loop for $i \geq 1$ (as the running time for $i = 0$ does not affect the final bound). Furthermore, as noted there, we have that $t \cdot |J_i| = O(n^{\delta_{\text{out}}})$ at each iteration of the loop. I.e., choosing $\eta_1 > 0$ so that $t = 2^i = O(n^{\eta_1})$, we have that $|J_i| \leq O(n^{\eta_2})$ where $\eta_2 \leq \min\{1, \delta_{\text{out}} - \eta_1\}$. We then have that $m = O(tn^{\alpha}) = O(n^{\eta_1+\alpha})$ for an arbitrarily small constant $\alpha > 0$ to be set in the analysis. We note that $T_{\mathcal{M}}(n,t) \leq \widetilde{O}(n)$. Moreover, we have that computing F takes $|J_i| \cdot T_{\mathcal{R}}(n,t) \leq n^{\delta_{\text{out}}-\eta_1} \cdot t^{1+\alpha} \cdot \text{poly}(\log n) = \widetilde{O}(n^{\delta_{\text{out}}+\alpha})$ time.

We next sketch how to modify the algorithm in the proof of Lemma 16 to run in $\widetilde{O}(n^{\delta_{\text{in}}})$ time. We note that the matrix X sampled there has $O(\log n)$ rows, and that we need to compute XAB' and XF. Computing XA and $(XA)B'$ takes $\widetilde{O}(n^{\delta_{\text{in}}})$ time by Proposition 9. On the other hand, we have that $\mathcal{R}$ runs in $t^{1+\alpha} \cdot \text{poly}(\log n) = \widetilde{O}(n^{\eta_1+\alpha})$ time, and therefore each of the $|J_i| = O(n^{\delta_{\text{out}}-\eta_1})$ columns of F must be $\widetilde{O}(n^{\eta_1+\alpha})$ sparse. Therefore, we can compute XF in time $\widetilde{O}(n^{\delta_{\text{out}}+\alpha})$ time.

It remains to analyze the time complexity of computing $D = HAB'$, since all other operations are efficient. We have that $A' := HA$ can be computed in $\widetilde{O}(n^{\delta_{\text{in}}})$ time by Theorem 13. Furthermore, since H is $\text{poly}(\log n)$ column sparse, A' is itself $\widetilde{O}(n^{\delta_{\text{in}}})$ sparse. It follows that $HAB' = A'B'$ can also be computed in $O(n^{\omega_{\delta_{\text{in}}}(\eta_1,1,\eta_2)+\varepsilon})$ time for any constant $\varepsilon > 0$.

To finish the proof, we note that the for loop uses at most $O(\log n)$ iterations, and then bound the running time of each iteration. The running time of the ith iteration is bounded from above by $\widetilde{O}(n^{\delta_{\text{out}}+\alpha} + n^{\delta_{\text{in}}} + n^{\omega_{\delta_{\text{in}}}(\eta_1,1,\eta_2)+\varepsilon})$ for arbitrarily small constants α and ε. In particular, to prove the claim, it suffices to show that both δ_{out} and $\omega_{\delta_{\text{in}}}(\eta_1, 1, \eta_2)$ are bounded from above by β. First, by the definition of β, we have that $\omega_{\delta_{\text{in}}}(\eta_1, 1, \eta_2) \leq \beta$.

Second, by Proposition 11, Item 2, we have that

$$\beta \geq \omega_{\delta_{\text{in}}}(\eta^*, 1, \delta_{\text{out}} - \eta^*) \geq \delta_{\text{out}} \ ,$$

where $\eta^* := \min\{1, \delta_{\text{out}}\}$. Indeed, Proposition 11, Item 2 applies because of the following bounds. First, $\eta^* = \min\{1, \delta_{\text{out}}\} \leq 1 \leq \delta_{\text{in}}$. Second, if $\delta_{\text{out}} < 1$, then $\delta_{\text{out}} - \eta^* = \delta_{\text{out}} - \delta_{\text{out}} = 0 \leq \delta_{\text{in}}$. Third, if $\delta_{\text{out}} \geq 1$, then $\delta_{\text{out}} - \eta^* = \delta_{\text{out}} - 1 \leq 2 - 1 = 1 \leq \delta_{\text{in}}$. Here we have repeatedly used the assumption that $\delta_{\text{in}} \geq 1$. ◀

We recover Theorem 2 from Theorem 19 by setting $\delta_{\text{in}} = 2$ and noting that for $\delta_{\text{out}} \geq 1$,

$$\max_{\substack{\eta_1,\eta_2\in[0,1]\,,\\ \eta_1+\eta_2=\delta_{\text{out}}}} \omega_{\delta_{\text{in}}}(\eta_1, 1, \eta_2) = \max_{\substack{\eta_1,\eta_2\in[0,1]\,,\\ \eta_1+\eta_2=\delta_{\text{out}}}} \omega(\eta_1, 1, \eta_2) \leq \omega(\delta_{\text{out}} - 1, 1, 1)$$

by Corollary 8.

While we state Theorem 19 for $\delta_{\text{in}} \geq 1$ for simplicity, we remark that the proof can be extended to the case of $\delta_{\text{in}} \in [0, 2]$. The main challenge in this case is that we cannot afford to run the algorithm $\mathcal{M}$ it time $\widetilde{O}(n)$ anymore (as this bound might be larger than the claimed upper bound on the running time of the algorithm). To overcome this, we note that the construction of the matrix H in our compressed sensing scheme (presented in full in an appendix of the full version of this paper [6]) is in fact strongly explicit. That is, one can compute all non-zero entries of a given column of H, and therefore multiply by H efficiently without even running the algorithm $\mathcal{M}$ first.

4.3 Optimal FSMM (and OSMM)

We now prove that our randomized algorithm for FSMM in Theorem 19 (which is equivalent to [1, Lemma 3.1]) is optimal in the sense that any improvement on it would imply a corresponding improvement on input-sparse rectangular matrix multiplication exponents. In fact, we show an *equivalence* between these exponents and the running time exponent for $(\delta_{\text{in}}, \delta_{\text{out}})$-FSMM$_R$. We also show an analogous result for the optimality of the algorithm in Theorem 2 for OSMM.

▶ **Proposition 20.** *Let $\delta_{in} \in [1, 2]$, $\delta_{out} \in [0, 2\delta_{in}]$, let R be a ring, and let $\beta \geq 0$. Then there is a randomized algorithm for $(\delta_{in}, \delta_{out})$-FSMM$_R$ that runs in time $O(n^{\beta+\varepsilon})$ for every constant $\varepsilon > 0$ if and only if $\beta \geq \max\{\delta_{in}, \omega_{\delta_{in}}(\eta_1, 1, \eta_2)\}$ for all $\eta_1, \eta_2 \in [0, 1]$ with $\eta_1 + \eta_2 = \delta_{out}$.*

Furthermore, if $\delta_{out} \in [1, 2]$ then there is a randomized algorithm for δ_{out}-OSMM that runs in time $O(n^{\beta+\varepsilon})$ for every constant $\varepsilon > 0$ if and only if $\beta \geq \omega(\delta_{out} - 1, 1, 1)$.

Proof. Let $\eta_1, \eta_2 \in [0, 1]$ with $\eta_1 + \eta_2 = \delta_{\text{out}}$. We reduce the problem of computing the product of $A \in R^{m\times n}$ and $B \in R^{n\times p}$ for $m = O(n^{\eta_1})$ and $p = O(n^{\eta_2})$ when A and B are both $O(n^{\delta_{\text{in}}})$-sparse to $(\delta_{\text{in}}, \delta_{\text{out}})$-FSMM$_R$. The reduction simply outputs $A' := (A^T, \mathbf{0}, \ldots, \mathbf{0})^T \in R^{n\times n}$ and $B' := (B, \mathbf{0}, \ldots, \mathbf{0}) \in R^{n\times n}$. I.e., A' and B' are simply A and B padded with enough rows and columns of 0s, respectively, to make them square. It is easy to recover the product AB from the product $A'B'$. Furthermore, $\|A'\|_0 = \|A\|_0 = O(n^{\delta_{\text{in}}})$, $\|B'\|_0 = \|B\|_0 = O(n^{\delta_{\text{in}}})$, and, because $AB \in R^{m\times p}$, $\|A'B'\|_0 \leq mp \leq O(n^{\eta_1+\eta_2}) = O(n^{\delta_{\text{out}}})$, as needed. Because the

reduction worked for arbitrary $\eta_1, \eta_2 \in [0,1]$ with $\eta_1 + \eta_2 = \delta_{\text{out}}$, we get that if there is an algorithm using $O(n^{\beta+\varepsilon})$ operations over R for $(\delta_{\text{in}}, \delta_{\text{out}})$-FSMM$_R$ for some $\beta > 0$ and any $\varepsilon > 0$, then $\omega_{\delta_{\text{in}}}(\eta_1, 1, \eta_2) \le \beta$. Furthermore, we trivially have that such β must satisfy $\beta \ge \delta_{\text{in}}$ since it takes $\Omega(n^{\delta_{\text{in}}})$ time to read an $(\delta_{\text{in}}, \delta_{\text{out}})$-FSMM$_R$ instance.

On the other hand, Theorem 19 implies that if $\beta \ge \delta_{\text{in}}$ and $\beta \ge \omega_{\delta_{\text{in}}}(\eta, 1, \delta_{\text{out}} - \eta)$ for all $\eta \in [0, \delta_{\text{out}}]$ then $(\delta_{\text{in}}, \delta_{\text{out}})$-FSMM$_R$ is solvable using $O(n^{\beta+\varepsilon})$ many operations over R.

The result for OSMM follows by combining the padding argument above, instantiated with $\delta_{\text{in}} = 2$ and $\eta = 1$, and Theorem 2. ◀

References

1 Amir Abboud, Karl Bringmann, Nick Fischer, and Marvin Künnemann. The time complexity of fully sparse matrix multiplication. In *SODA*, 2024. 1, 2, 3, 4, 5, 7, 10, 17, 18

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *SODA*, 2025. 2, 9

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *SODA*, 2021. 9

4 Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *ICDT*, 2009. 3

5 Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Matrix multiplication verification using coding theory. In *RANDOM*, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.42. 4, 6

6 Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Output-sparse matrix multiplication using compressed sensing, 2025. Full version. Available at arXiv:2508.10250. doi:10.48550/arXiv.2508.10250. 1, 3, 8, 11, 18

7 Radu Berinde, Anna C. Gilbert, Piotr Indyk, Howard Karloff, and Martin J. Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *Allerton*, 2008. 3, 7, 8, 11, 12

8 Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. In *ICTCS*, 2015. 3

9 Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006. doi:10.1109/TIT.2005.862083. 8

10 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In *SIGMOD*, 2020. 3

11 Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *FOCS*, 2023. 9

12 Rūsiņš Freivalds. Fast probabilistic algorithms. In *MFCS*, 1979. 8, 13

13 Jianhua Gao, Weixing Ji, Fangli Chang, Shiyu Han, Bingxin Wei, Zeming Liu, and Yizhuo Wang. A systematic survey of general sparse matrix-matrix multiplication. *ACM Computing Surveys*, 55(12):1–36, 2023. doi:10.1145/3571157. 3

14 Leszek Gasieniec, Christos Levcopoulos, Andrzej Lingas, Rasmus Pagh, and Takeshi Tokuyama. Efficiently correcting matrix products. *Algorithmica*, 79(2):428–443, 2017. doi:10.1007/S00453-016-0202-3. 3, 6

15 Anna C. Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010. doi:10.1109/JPROC.2010.2045092. 8

16 John R. Gilbert, Steve Reinhardt, and Viral B. Shah. High-performance graph algorithms from parallel sparse matrices. In *PARA*, 2006. 3

17 Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *Journal of the ACM*, 56(4):1–34, 2009. doi:10.1145/1538902.1538904. 1, 3, 7, 8, 11

18 Fred G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software*, 4(3):250–269, 1978. doi:10.1145/355791.355796. 3, 10

19 Piotr Indyk. Explicit constructions for compressed sensing of sparse signals. In *SODA*, 2008. 5

20 Mark A. Iwen and Craig V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Information Processing Letters*, 109(10):468–471, 2009. doi:10.1016/J.IPL.2009.01.010. 3, 5

21 Riko Jacob and Morten Stöckel. Fast output-sensitive matrix multiplication. In *ESA*, 2015. 3

22 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *SoCG*, 2006. 7, 10, 11

23 Marvin Künnemann. On nondeterministic derandomization of Freivalds' algorithm: Consequences, avenues and algorithmic progress. In *ESA*, 2018. doi:10.4230/LIPIcs.ESA.2018.56. 3, 4, 5

24 Konstantin Kutzkov. Deterministic algorithms for skewed matrix products. In *STACS*, 2013. doi:10.4230/LIPIcs.STACS.2013.466. 3, 4, 5

25 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *FOCS*, 2012. 10

26 François Le Gall. Faster rectangular matrix multiplication by combination loss analysis. In *SODA*, 2024. 9

27 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *SODA*, 2018. 9

28 Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. In *ESA*, 2009. 3

29 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 23(2):171–185, 1983. doi:10.1016/0304-3975(83)90054-3. 9, 10

30 Rasmus Pagh. Compressed matrix multiplication. In *ITCS*, 2012. 3

31 Rasmus Pagh and Morten Stöckel. The input/output complexity of sparse matrix multiplication. In *ESA*, 2014. 8

32 Daniel S. Roche. What can (and can't) we do with sparse polynomials? In *ISSAC*, 2018. 3

33 Oguz Selvitopi, Md Taufique Hussain, Ariful Azad, and Aydın Buluç. Optimizing high performance Markov clustering for pre-exascale architectures. In *IPDPS*, 2020. 3

34 Dirk Van Gucht, Ryan Williams, David P. Woodruff, and Qin Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *PODS*, 2015. 3

35 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *SODA*, 2024. 3, 5, 9

36 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions On Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466. 3, 7, 10, 11

37 Yudong Zhang, Lenan Wu, Geng Wei, and Shuihua Wang. A novel algorithm for all pairs shortest path problem based on matrix multiplication and pulse coupled neural network. *Digital Signal Processing*, 21(4):517–521, 2011. doi:10.1016/J.DSP.2011.02.004. 3