Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 162 (2021)

Fast, Algebraic Multivariate Multipoint Evaluation in Small Characteristic and Applications

Vishwas Bhargava* Sumanta Ghosh† Mrinal Kumar‡
Chandra Kanta Mohapatra‡

Abstract

Multipoint evaluation is the computational task of evaluating a polynomial given as a list of coefficients at a given set of inputs. Besides being a natural and fundamental question in computer algebra on its own, fast algorithms for this problem are also closely related to fast algorithms for other natural algebraic questions like polynomial factorization and modular composition. And while *nearly linear time* algorithms have been known for the univariate instance of multipoint evaluation for close to five decades due to a work of Borodin and Moenck [BM74], fast algorithms for the multivariate version have been much harder to come by. In a significant improvement to the state of art for this problem, Umans [Uma08] and Kedlaya & Umans [KU11] gave nearly linear time algorithms for this problem over field of small characteristic and over all finite fields respectively, provided that the number of variables n is at most $d^{o(1)}$ where the degree of the input polynomial in every variable is less than d. They also stated the question of designing fast algorithms for the large variable case (i.e. $n \notin d^{o(1)}$) as an open problem.

In this work, we show that there is a deterministic algorithm for multivariate multipoint evaluation over a field $\mathbb{F}_q$ of characteristic p which evaluates an n-variate polynomial of degree less than d in each variable on N inputs in time

$$\left((N + d^n)^{1+o(1)} \operatorname{poly}(\log q, d, n, p)\right),$$

provided that p is at most $d^{o(1)}$, and q is at most $(\exp(\exp(\exp(\cdots(\exp(d))))))$, where the height of this tower of exponentials is fixed. When the number of variables is large (e.g. $n \notin d^{o(1)}$), this is the first nearly linear time algorithm for this problem over any (large enough) field.

Our algorithm is based on elementary algebraic ideas and this algebraic structure naturally leads to the following two independently interesting applications.

*Department of Computer Science, Rutgers University, Piscataway, NJ 08854. Research supported in part by the Simons Collaboration on Algorithms and Geometry and NSF grant CCF-1909683. vishwas1384@gmail.com.

†A part of this work was done during a postdoctoral stay at the Department of Computer Science & Engineering, IIT Bombay, Mumbai, India. besusumanta@gmail.com.

‡Department of Computer Science & Engineering, IIT Bombay, Mumbai, India. mrinal|ckm@cse.iitb.ac.in.

ISSN 1433-8092

- We show that there is an *algebraic* data structure for univariate polynomial evaluation with nearly linear space complexity and sublinear time complexity over finite fields of small characteristic and quasipolynomially bounded size. This provides a counterexample to a conjecture of Milterson [Mil95] who conjectured that over small finite fields, any algebraic data structure for polynomial evaluation using polynomial space must have linear query complexity.
- We also show that over finite fields of small characteristic and quasipolynomially bounded size, Vandermonde matrices are not rigid enough to yield size-depth tradeoffs for linear circuits via the current quantitative bounds in Valiant's program [Val77]. More precisely, for every fixed prime p, we show that for every constant $\varepsilon > 0$, and large enough n, the rank of any $n \times n$ Vandermonde matrix V over the field $\mathbb{F}_{p^a}$ can be reduced to $\left(n/\exp(\Omega(\text{poly}(\varepsilon)\sqrt{\log n}))\right)$ by changing at most $n^{\Theta(\varepsilon)}$ entries in every row of V, provided $a \leq \text{poly}(\log n)$. Prior to this work, similar upper bounds on rigidity were known only for special Vandermonde matrices. For instance, the Discrete Fourier Transform matrices and Vandermonde matrices with generators in a geometric progression [DL20].

Contents

1 Introduction

We study the question of designing fast algorithms for the following very natural and fundamental computational task.

Question 1.1 (Multipoint Evaluation)**.** *Given the coefficient vector of an n-variate polynomial f of degree at most $d-1$ in each variable over a field $\mathbb{F}$ and a set of points $\{\boldsymbol{\alpha}_i : i \in [N]\}$ in $\mathbb{F}^n$, output $f(\boldsymbol{\alpha}_i)$ for each $i \in [N]$.*

Besides being a natural and fundamental question in computer algebra on its own, fast algorithms for this problem is also closely related to fast algorithms for other natural algebraic questions like polynomial factorization and modular composition [KU11].

The input for this question can be specified by $(d^n + Nn)$ elements of $\mathbb{F}$ and clearly, there is a simple algorithm for this task which needs roughly $((d^n \cdot N)\,\mathrm{poly}(n,d))$ arithmetic operations over $\mathbb{F}$: just evaluate f on $\boldsymbol{\alpha}_i$ for every i iteratively. Thus for $N = d^n$, the number of field operations needed by this algorithm is roughly quadratic in the input size. While *nearly linear time*[1] algorithms have been known for the univariate instance of multipoint evaluation [BM74] for close to five decades, fast algorithms for the multivariate version have been much harder to come by. In a significant improvement to the state of art for this problem, Umans [Uma08] and Kedlaya & Umans [KU11] gave nearly linear time algorithms for this problem over fields of small characteristic and over all finite fields respectively, provided that the number of variables n is at most $d^{o(1)}$ where the degree of the input polynomial in every variable is less than d. They also stated the question of designing fast algorithms for the large variable case (i.e. $n \notin d^{o(1)}$) as an open problem.

In this work, we make some concrete progress towards this question over finite fields of small characteristic (and not too large size). We also show two independently interesting applications of our algorithm. The first is to an upper bound for algebraic data structures for univariate polynomial evaluation over finite fields and second is to an upper bound on the rigidity of Vandermonde matrices over fields of small characteristic. Before stating our results, we start with a brief outline of each of these problems and discuss some of the prior work and interesting open questions. We state our results in section 2.

1.1 Algorithms for multivariate multipoint evaluation

For the case of univariate polynomials and $N = d$, Borodin and Moenck [BM74] showed that multipoint evaluation can be solved in $O(d\,\mathrm{poly}(\log d))$ field operations via a clever use of the Fast Fourier Transform (FFT).

For multivariate polynomials, when the evaluation points of interest are densely packed in a product set in $\mathbb{F}^n$, FFT based ideas naturally generalize to multivariate multipoint evaluation

[1] Throughout this paper, we use the phrase "nearly linear time" to refer to algorithms such that for all sufficiently large m, they run in time $m^{1+o(1)}$ on inputs of size m.

yielding a nearly linear time algorithm. However, if the evaluation points are arbitrary and the underlying field is sufficiently large[2], and in particular not packed densely in a product set, the question of designing algorithms for multipoint evaluation that are significantly faster than the straightforward quadratic time algorithm appears to be substantially harder. In fact, the first significant progress in this direction was achieved nearly three decades after the work of Borodin and Moenck by Nüsken and Ziegler [NZ04] who showed that for $n = 2$ and $N = d^2$, multipoint evaluation can be solved in most $O(d^{\omega_2/2+1})$ operations, where ω_2 is the exponent for multiplying a $d \times d$ and a $d \times d^2$ matrix. The algorithm in [NZ04] also generalizes to give an algorithm for general n that requires $O(d^{\omega_2/2(n-1)+1})$ field operations.[3] Two significant milestones in this line of research are the results of Umans [Uma08] and Kedlaya & Umans [KU11] who designed nearly linear time algorithms for this problem for fields of small characteristic and over all finite fields respectively, provided the number of variables n is at most $d^{o(1)}$. We now discuss these results in a bit more detail.

Umans [Uma08] gave an algorithm for multipoint evaluation over finite fields of small characteristic. More precisely, the algorithm in [Uma08] solves multipoint evaluation in time $O((N + d^n)(n^2 p)^n) \cdot \mathrm{poly}(d, n, p, \log N)$ over a finite field $\mathbb{F}$ of characteristic p. Thus, when p and n are $d^{o(1)}$, the running time can be upper bounded by $(N + d^n)^{1+\delta}$ for every constant $\delta > 0$ and d, N sufficiently large. In addition to its impressive running time, the algorithm of Umans [Uma08] is also algebraic, i.e. it only requires algebraic operations over the underlying field. With multipoint evaluation naturally being an algebraic computational problem, an algebraic algorithm for it has some inherent aesthetic appeal. The results in [Uma08], while being remarkable has two potential avenues for improvement, namely, a generalization to other fields and to the case when the number of variables is not $d^{o(1)}$.

In [KU11], Kedlaya & Umans addressed the first of these issues. They showed that multipoint evaluation can be solved in nearly linear time over *all* finite fields. More precisely, for every $\delta > 0$, their algorithm for multipoint evaluation has running time $(N + d^n)^{1+\delta} \log^{1+o(1)} q$ over any finite field $\mathbb{F}$ of size q, provided d is sufficiently large and $n = d^{o(1)}$. Quite surprisingly, the algorithm in [KU11] is not algebraic. It goes via lifting the problem instance from the finite field $\mathbb{F}$ to an instance over $\mathbb{Z}$ and then relies on an extremely clever and unusual application of the Chinese Remainder Theorem to reduce the instance over $\mathbb{Z}$ back to instances over small finite fields. Intuitively, the gain in the entire process comes from the fact that in the reduced instances obtained over small finite fields, the evaluation points of interests are quite densely packed together inside a small product set and a standard application of the multidimensional FFT can be used to solve these small field instances quite fast. Another closely related result is a recent work of Björklund,

[2]Over small fields, for instance if $|\mathbb{F}| \leq d^{1+o(1)}$ or $|\mathbb{F}|^n \leq N^{1+o(1)}$, a standard application of multidimensional Fast Fourier Transform which just evaluates the polynomial at all points in $\mathbb{F}^n$ and looks up the values at the N input points works in nearly linear time. So, throughout the discussion on multipoint evaluation, we assume that $\mathbb{F}$ is large enough.

[3]The results in both [BM74] and [NZ04] work for arbitrary N, but for simplicity have been stated for $N = d$ and $N = d^2$ respectively here.

Kaski and Williams [BKW19] who (among other results) give an algorithm for multivariate multipoint evaluation but their time complexity depending polynomially on the field size (and not polynomially on the logarithm of the field size).

In addition to these algorithms for multivariate multipoint evaluation, Umans [Uma08] and Kedlaya & Umans [KU11] also show that these fast algorithms lead to significantly faster than previously known algorithms for many other natural algebraic problems. This includes the questions of modular composition where the input consists of three univariate polynomials $f, g, h \in \mathbb{F}[X]$ of degree less than d each and the goal is to output $(f(g(X)) \mod h(X))$. In addition to being interesting on its own, faster algorithms for modular composition over finite fields are known to directly imply faster algorithms for univariate polynomial factorization over such fields. Indeed, using their nearly linear time algorithm for multipoint evaluation , Umans [Uma08] and Kedlaya & Umans [KU11] obtain the currently fastest known algorithms for univariate polynomial factorization over finite fields. We refer the reader to [KU11] for a detailed discussion of these connections and implications.

In spite of the significant progress on the question of algorithms for multipoint evaluation in [Uma08] and [KU11], some very natural related questions continue to remain open. For instance, we still do not have nearly linear time algorithms for multipoint evaluation when the number of variables is large, e.g. $n \notin d^{o(1)}$ over any (large enough) finite field, or when the field is not finite. Since multipoint evaluation is quite naturally an algebraic computational problem, it would also be quite interesting to have a nearly linear size arithmetic circuits over the underlying field for this problem even if such a circuit cannot be efficiently constructed. Currently, small circuits of this kind are only known over finite fields of small characteristic due to the results in [Uma08]. The algorithm in [KU11] does not seem to yield such a circuit since it is not algebraic over the underlying field.

1.2 Data structures for polynomial evaluation

One particular implication of the results in [KU11] is towards the question of constructing efficient data structures for polynomial evaluation over finite fields. The *data* here is a univariate polynomial $f \in \mathbb{F}[X]$ of degree less than n over a finite field $\mathbb{F}$. The goal is to process this data and store it in a way that we can support fast polynomial evaluation queries, i.e. queries of the form: given an $\alpha \in \mathbb{F}$ output $f(\alpha)$. The two resources of interest here are the space required to store the data and the number of locations [4] accessed for every query, i.e the query complexity. There are two very natural solutions to this problem.

[4] This can be measured in terms of the cells accessed where each cell contains an element over the underlying field. This is an instance of the cell probe model and is quite natural in the context of algebraic data structures for algebraic problems. Alternatively, we can also measure the space and query complexity in terms of the number of bits stored and accessed respectively.

- We can store the coefficient vector of the polynomial f in the memory and for each query $\alpha \in \mathbb{F}$, we can read the whole memory to recover the coefficient vector of f and hence compute $f(\alpha)$. Thus, the space complexity and the query complexity of this data structure are both $(O(n \log q))$ bits, with clearly the space requirement being the best that we can hope for.

- The second natural data structure for this problem just stores the evaluation of f on all $\alpha \in \mathbb{F}$ in the memory, and on any query, can just read off the relevant value. Thus, the space complexity here is $O(q \log q)$ bits, but the query complexity is $O(\log q)$ bits (which is the best that we can hope for). For q being much larger than n the space requirement here is significantly larger than that in the first solution.

Using their algorithm for multipoint evaluation in [KU11], Kedlaya & Umans construct a data structure for this problem with space complexity $n^{1+\delta} \log^{1+o(1)} q$ and query complexity $\text{poly}(\log n) \cdot \log^{1+o(1)} q$ for all $\delta > 0$ and sufficiently large n. Thus, the space needed is quite close to optimal, and the query complexity is within a $\text{poly}(\log n)$ factor of the optimal. Quite surprisingly, this data structure is not algebraic since it relies on the multipoint evaluation algorithm in [KU11] which in turn relies on non-algebraic modular arithmetic. We also note that while the algorithm for multipoint evaluation over fields of small characteristic in [Uma08] is algebraic, to the best of our knowledge, it does not immediately yield a data structure for polynomial evaluation. We remark that while the discussion here has been focused on data structures for univariate polynomial evaluation, the ideas in [KU11] continue to work as it is even for the multivariate version of this problem and gives quantitatively similar results there. In fact, their solution to the univariate problem goes via a reduction to the multivariate case!

In a recent work, Björklund, Kaski and Williams [BKW19] also prove new data structures upper bounds for polynomial evaluations for multivariate polynomials over finite fields. These data structures are algebraic and are based on some very neat geometric ideas closely related to the notion of Kakeya sets over finite fields. Their construction can be viewed as giving a tradeoff in the space and query complexities but at least one of these parameters always appears to have polynomial dependence on the size of the underlying finite field. This is in contrast to the results in [KU11] where the query complexity depends nearly linearly on $\log q$ which is more desirable for this problem.

A very natural open question in this line of research is to obtain an algebraic data structure for this problem which matches the space and query complexity of the results in [KU11]. Currently, we do not have an algebraic data structure for this problem over with even polynomial space and sublinear query complexity over any sufficiently large field. In fact, Milterson [Mil95] showed that for algebraic data structures over finite fields of size exponential in n, if the space used is $\text{poly}(n)$, then the trivial data structure obtained by storing the given polynomial as a list of coefficients and reading off everything in the memory on every query is essentially the best we can do. Milterson

also conjectured a similar lower bound to hold over smaller fields. Thus, over smaller finite fields (for instance, finite fields of size $\text{poly}(n)$), either proving a lower bound similar to that in [Mil95] , or constructing *algebraic* data structures for polynomial evaluation with perform guarantees similar to those in [KU11] are extremely interesting open problems. For the later goal, it would be a good start to even have an algebraic data structure that does significantly better than the trivial solution of storing the coefficient vector of the given polynomial.

1.3 Non-rigidity of Vandermonde matrices

An application of our results for multipoint evaluation is towards upper bounds for the rigidity of Vandermonde matrices. In this section, we give a brief overview of matrix rigidity.

Let $\mathbb{F}$ be any field. An $n \times n$ matrix M over $\mathbb{F}$ is said to be (r, s) rigid for some parameters $r, s \in \mathbb{N}$ if M cannot be written as a sum of $n \times n$ matrices of rank at most r and sparsity at most s. In other words, the rank of M cannot be reduced to less than or equal to r by changing at most s of its entries. This notion was defined by Valiant [Val77] who showed that if the linear transformation given by M can be computed by an arithmetic circuit of size $O(n)$ and depth $O(\log n)$, then M is not $(O(n/\log\log n), O(n^{1+\varepsilon}))$ rigid for any $\varepsilon > 0$. For brevity, we say that a family of matrices is *Valiant rigid* if it is $(O(n/\log\log n), O(n^{1+\varepsilon}))$ rigid for some $\varepsilon > 0$. Thus, constructing an explicit family of matrices that are Valiant rigid suffices for proving superlinear lower bounds for log depth circuits for an explicit family of linear transformations; an extremely interesting problem that continues to be wide open. The progress on this question has been painfully slow although there have been several highly non-trivial and extremely interesting developments in this direction, e.g. [SSS97, Fri93, Lok00, Lok01, Lok06, GT18, AC19, BHPT20].

Even though the question of provable rigidity lower bounds for explicit matrix families has remained elusive, there has been a steady accumulation of various families of explicit matrices that are suspected to be rigid. For instance, Hadamard Matrices, Design Matrices, the Discrete Fourier Transform (DFT) matrices and various Vandermonde Matrices have all been suspected to be rigid with varying parameters at various points in time. For some of these cases, we even have rigidity lower bounds either for special cases or with parameters weaker than what is needed for Valiant's connection to arithmetic circuit lower bounds. However, quite surprisingly Alman & Williams [AW17] showed that Hadamard matrices are not Valiant rigid over $\mathbb{Q}$. This result was succeeded by a sequence of recent results all showing that many more families of matrices suspected to be highly rigid are in fact not Valiant rigid. This includes the work of Dvir & Edelman [DE19], the results of Dvir & Liu [DL20], those of Alman [Alm21] and Kivva [Kiv21]. This list of suspected to be highly rigid that have since been proven innocent includes families like Hadamard Matrices [AW17], Discrete Fourier Transform (DFT) Matrices, Circulant and Toeplitz matrices [DL20] and any family of matrices that can be expressed as a Kronecker product of small matrices [Alm21, Kiv21].

However, a notable family of matrices missing from this list is that of Vandermonde matrices. Special cases of Vandermonde matrices, for instance the DFT matrices, are known to be not be Valiant rigid, and in fact this result extends to the case of all Vandermonde matrices where the *generators* are in geometric progression.[5] However, the case of Vandermonde matrices with arbitrary generators is still not well understood.[6]

2 Our results

We now state our results formally and try to place them in the context of prior work.

2.1 Algorithms for multivariate multipoint evaluation

Our main result is a fast algebraic algorithm for multipoint evaluation over fields of small characteristic. We state this result informally here, and refer the reader to Theorem 7.1 for a formal statement.

Theorem 2.1 (Informal)**.** *Over a field $\mathbb{F}_{p^a}$ of characteristic p, there is a deterministic algorithm which evaluates a given n-variate polynomial of degree less than d in each variable on N inputs in time*

$$\left((N+d^n)^{1+o(1)} \cdot \mathrm{poly}(a,d,n,p)\right),$$

provided that p is at most $d^{o(1)}$ and a is at most $(\exp(\exp(\exp(\cdots(\exp(d))))))$, where the height of this tower of exponentials is fixed.

A few remarks are in order.

Remark 2.2. *Throughout this paper, we assume that we are given a description of the field $\mathbb{F}_{q=p^a}$ as a part of the input. For instance, we are given an irreducible polynomial $v(Y) \in \mathbb{F}_p[Y]$ of degree equal to $\log_p q$ and $\mathbb{F}_q \equiv \mathbb{F}_p[Y]/\langle v(Y)\rangle$.* ⌟

Remark 2.3. *Our algorithms for Theorem 2.1 can be viewed as naturally giving an arithmetic circuit of nearly linear size for multivariate multipoint evaluation over the underlying finite field $\mathbb{F}_{p^a}$. Throughout this paper, this is what we mean when we say we have an "algebraic" algorithm. Moreover, given a description of $\mathbb{F}_q$ as in Remark 2.2, we can use the algorithm in Theorem 2.1 to output such a circuit for multipoint evaluation in nearly linear time.*

⌟

As alluded to in the introduction, when the number of variables is large (e.g. $n \notin d^{o(1)}$), this is the first nearly linear time algorithm for this problem over any sufficiently large field. Prior to this

[5] An $n \times n$ Vandermonde matrix over a field $\mathbb{F}$ is specified by a list of n field elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ in $\mathbb{F}$ that we call generators. The rows and columns are indexed by $\{0, 1, \ldots, n-1\}$ and the (i,j)th entry of the matrix equals ${\alpha_i}^j$.

[6] Lokam [Lok00] shows that $n \times n$ Vandermonde matrices with algebraically independent generators are at least $(\sqrt{n}, \Omega(n^2))$ rigid. This bound, however, is not sufficient for Valiant's program.

work, the fastest known algorithms for multivariate multipoint evaluation are due to the results of Umans [Uma08] and Kedlaya & Umans [KU11] who give nearly linear time algorithms for this problem over finite fields of small characteristic and all finite fields respectively when the number of variables n is at most $d^{o(1)}$. Theorem 2.1 answers an open question of Kedlaya & Umans [KU11] over the fields where it applies. For a comparison between our result, [Uma08] and [KU11] see Table 1. Here, $\mathrm{Char}(\mathbb{F})$ denotes the characteristic of the field $\mathbb{F}$.

Multivariate Multipoint evaluation over a finite field $\mathbb{F}_{q=p^a}$				
Results	Number of Operations	Algorithm type	Field constraint	Variable
[Uma08]	$(N+d^n)^{1+\delta}$, $\forall \delta > 0$, $\mathbb{F}_q$-operations	Algebraic	$\mathrm{Char}(\mathbb{F}_q) \leq d^{o(1)}$	$n \leq d^{o(1)}$
[KU11]	$(N + d^n)^{1+\delta} \log^{1+o(1)} q$, $\forall \delta > 0$, many bit operations	Non-algebraic	over any finite field	$n \leq d^{o(1)}$
This work	$(N + d^n)^{1+o(1)} \cdot \mathrm{poly}(d, n, \mathrm{Char}(\mathbb{F}_q), \log q)$ $\mathbb{F}_q$-operations (Theorem 7.1)	Algebraic	$\mathrm{Char}(\mathbb{F}_q) \leq d^{o(1)}$, $q \leq (\exp(\exp(\cdots(\exp(d)))))$, where the height of this tower of exponentials is fixed	over any n

Table 1: Comparison with known results

By a direct connection between the complexity of multipoint evaluation and modular composition shown by Kedlaya & Umans [KU11], Theorem 2.1 implies a nearly linear time algorithm for modular composition even when the number of variables n is not less than $d^{o(1)}$. In [KU11], such an algorithm was obtained when $n < d^{o(1)}$ (over all finite fields). More precisely, we have the following corollary.

Corollary 2.4 (Informal)**.** *Let $\mathbb{F}_{p^a}$ be a field of characteristic p. Then, there is an algorithm that on input an n-variate polynomial $f(X_1, X_2, \ldots, X_n)$ of individual degree less than d and univariate polynomials $g_1(X), \ldots, g_n(X)$ and $h(X)$ in $\mathbb{F}_{p^a}[X]$ with degree less than N, outputs the polynomial*

$$f(g_1(X), g_2(X), \ldots, g_n(X)) \mod h(X)$$

in time

$$(d^n + N)^{1+o(1)} \cdot \mathrm{poly}(a, d, n, p)\,,$$

provided that p is at most $d^{o(1)}$ and a is at most $(\exp(\exp(\exp(\cdots(\exp(d))))))$, where the height of this tower of exponentials is fixed.

Our algorithm is based on elementary algebraic ingredients. One of these ingredients is the basic fact that the restriction of a low degree multivariate polynomial to a low degree curve is a low degree univariate polynomial! We use this fact together with some other algebraic tools, e.g. univariate polynomial interpolation (with multiplicities), structure of finite fields, and multidimensional FFT for our algorithm. We describe an overview of the main ideas in the proof in section 3. We also note that even though the algorithm in [Uma08] is algebraic, it appears to be based on ideas very different from those in this paper. In particular, Umans relies on a clever reduction from the multivariate problem to the univariate problem by working over appropriate extension of the underlying field. This is then combined with the classical univariate multipoint evaluation algorithm to complete the picture. Our algorithm, on the other hand, does not involve a global reduction from the multivariate set up to the univariate set up, and crucially relies on more local properties of low degree multivariate polynomials.

Another related prior work is a result of Björklund, Kaski and Williams [BKW19], who give a data structure (and an algorithm) for multipoint evaluation and some very interesting consequences to fast algorithms for problems in #P. We note that at a high level, the structure of our algorithm is similar to that of the algorithm of Björklund, Kaski and Williams [BKW19]. However, the technical details and quantitative bounds achieved are different. One major difference is that the time complexity of the algorithm in [BKW19] depends *polynomially* on the field size. Thus strictly speaking, with the field size growing, this algorithm is not polynomial time in the input size. On the other hand, the time complexity of the algorithms in the works of Umans [Uma08], Kedlaya & Umans [KU11] and that in Theorem 2.1 depends polynomially in the logarithm of the field size, as is more desirable. We discuss the similarities and differences in the high level structure of the algorithm in [BKW19] and that in Theorem 2.1 in a little more detail in section 3.

2.2 Data structures for polynomial evaluation

As an interesting application of our ideas in Theorem 2.1, we get the following upper bound for data structure for polynomial evaluation.

Theorem 2.5 (Informal)**.** *Let p be a fixed prime. Then, for all sufficiently large $n \in \mathbb{N}$ and all fields $\mathbb{F}_{p^a}$ with $a \leq \mathrm{poly}(\log n)$, there is an algebraic data structure for polynomial evaluation for univariate polynomials of degree less than n over $\mathbb{F}_{p^a}$ that has space complexity at most $n^{1+o(1)}$ and query complexity at most $n^{o(1)}$.*

A more precise version of Theorem 2.5 can be found in Theorem 8.1. We remark that by an algebraic data structure, we mean that there is an algebraic algorithm (in the spirit of Remark 2.3) over $\mathbb{F}_{p^a}$ that, when given the coefficients of a univariate polynomial f of degree at most n as input outputs the data structure $\mathcal{D}_f$ in time $n^{1+o(1)}$ and another algebraic algorithm which when given an $\alpha \in \mathbb{F}_{p^a}$ and query access to $\mathcal{D}_f$ outputs $f(\alpha)$ in time $n^{o(1)}$. In other words, there is an arithmetic

circuit C_1 over $\mathbb{F}_{p^a}$ with $n^{1+o(1)}$ outputs that when given the coefficients of f as input, outputs $\mathcal{D}_f$ and an arithmetic circuit C_2 with $n^{o(1)}$ inputs satisfying the following: for every $\alpha \in \mathbb{F}_{p^a}$, there is a subset $S(\alpha)$ of cells in $\mathcal{D}_f$ such that on input α and $\mathcal{D}_f|_{S(\alpha)}$, C_2 outputs $f(\alpha)$.

As alluded to in the introduction, Milterson [Mil95] showed that over finite fields that are exponentially large (in the degree parameter n), any algebraic data structure for polynomial evaluation with space complexity $\text{poly}(n)$ must have query complexity $\Omega(n)$. He also conjectured that the lower bound continues to hold over smaller fields.[7] Theorem 2.5 provides a counterexample to this conjecture when the underlying field has small characteristic and is quasipolynomially bounded in size.

The data structure of Kedlaya & Umans [KU11] outperforms the space and query complexities of the data structure in Theorem 2.5. However, their construction is not algebraic; essentially because their algorithm for multipoint evaluation is not algebraic.[8] However, their construction works over all finite fields, while we require fields of small characteristic that are quasipolynomially bounded in size. Umans' [Uma08] algorithm for multipoint evaluation on the other hand is algebraic, although to the best of our knowledge, this is not known to give a data structure for polynomial evaluation. Finally, we note that for the algebraic data structure in the work of Björklund, Kaski and Williams [BKW19], either the query complexity or the space complexity has polynomial dependence on the field size and thus even over fields of polynomial size it does not appear to give nearly linear space complexity or sublinear query complexity. However, the results in [BKW19] are stated for multivariate polynomials and it is not clear to us if for the special case of univariate polynomial one can somehow bypass this polynomial dependence on field size by a careful modification of their construction.

2.3 Upper bound on the rigidity of Vandermonde matrices

As the second application of the ideas in Theorem 2.1, we show the following upper bound on the rigidity of general Vandermonde matrices.

Theorem 2.6 (Informal)**.** *Let p be a fixed prime. Then, for all constants ε with $0 < \varepsilon < 0.01$ and for all sufficiently large n, if V is an $n \times n$ Vandermonde matrix over the field $\mathbb{F}_{p^a}$ for $a \leq \text{poly}(\log n)$, then the rank of V can be reduced to $\frac{n}{\exp\left(\Omega(\varepsilon^7 \log^{0.5} n)\right)}$ by changing at most $n^{1+\Theta(\varepsilon)}$ entries of V.*

For a more formal version of Theorem 2.6, we refer to Theorem 9.6. Theorem 2.6 extends the list of natural families of matrices that were considered potential explicit candidates for rigidity but turn out to not be rigid enough for Valiant's program [Val77] of obtaining size-depth tradeoffs

[7] We note that Milterson did not precisely quantify what *smaller* fields mean, but the case when the field size is a large polynomial in the degree parameter n is a natural setting, since the trivial data structures in this case do not have both nearly linear space and sublinear query complexity. Theorem 2.5 provides such a construction when the underlying field additionally has a small characteristic.

[8] This is also the reason why the data structure in [KU11] does not give a counterexample to Milterson's conjecture.

for linear arithmetic circuits via rigidity. Prior to this work, such upper bounds on rigidity were only known for special Vandermonde matrices, for instance, the Discrete Fourier transform matrix and Vandermonde matrices with generators in geometric progression [DL20].

Our proof of Theorem 2.6 crucially relies on the results in [DL20] and combines these ideas with ideas in the proof of Theorem 2.1. We discuss these in more details in the next section.

3 An overview of the proofs

In this section we describe some detail, the main high level ideas of our proofs. We begin with a detailed overview of our algorithms for multipoint evaluation. We have three algorithms (section 5, section 6 and section 7) starting with the simplest one and each subsequent algorithm building upon the previous one with some new ideas. We start with the simplest one here.

3.1 A simple algorithm for multipoint evaluation

We start with some necessary notation. Let p be a prime and $\mathbb{F}_q$ be a finite field with $q = p^a$. Let $f \in \mathbb{F}_q[\mathbf{x}]$ be an n-variate polynomial of degree at most $d-1$ in every variable and for $i = 1, 2, \ldots, N$ let $\boldsymbol{\alpha}_i \in \mathbb{F}_q^n$ be points. The goal is to output the value of f at each of these points $\boldsymbol{\alpha}_i$. As is customary, we assume that the field $\mathbb{F}_q$ is given as $\mathbb{F}_p[Y]/\langle v(Y)\rangle$ for some degree a irreducible polynomial $v(Y) \in \mathbb{F}_p[Y]$. In Lemma 4.3, we observe that given the irreducible polynomial $v(Y) \in \mathbb{F}_p[Y]$ such that $\mathbb{F}_q = \mathbb{F}_p[Y]/\langle v(Y)\rangle$ and any $u \in \mathbb{F}_q$, we can efficiently compute the coefficients of the univariate polynomial over $F_p[Y]$ corresponding to u via arithmetic operations over $\mathbb{F}_q$. Therefore, for the rest of this discussion, we assume that every field element (in the coefficients of f and the coordinates of $\boldsymbol{\alpha}_i$) are explicitly given to univariate polynomials of degree at most $a-1$ in $\mathbb{F}_p[Y]$.

We start with a discussion of the simplest version of our algorithm before elaborating on the other ideas needed for further improvements. The formal guarantees for this version can be found in Theorem 5.1. The algorithm can be thought to have two phases, the preprocessing phase and the local computation phase.

Preprocessing phase. We start with a description of the preprocessing phase.

- **A subfield of appropriate size:** As the first step of the algorithm, we compute a natural number b such that $p^{b-1} \leq adn \leq p^b$. For the ease of this discussion, let us assume that b divides a, and thus $\mathbb{F}_{p^b}$ is a subfield of $\mathbb{F}_q = \mathbb{F}_{p^a}$. If b does not divide a, then we work in a field $\mathbb{F}_{p^c}$ that is a common extension of $\mathbb{F}_{p^a}$ and $\mathbb{F}_{p^b}$.

- **Evaluating f on $\mathbb{F}_{p^b}^n$:** We now use the standard multidimensional Fast Fourier Transform algorithm to evaluate f on all of $\mathbb{F}_{p^b}^n$. This algorithm runs in quasilinear time in the input

size, i.e. $\tilde{O}(d^n + (p^{bn}))$, where $\tilde{O}$ hides $\text{poly}(d, n, p, b)$ factors. From our choice of b, we note that this quantity is at most $\tilde{O}((padn)^n)$.

Local computation phase. We now describe the local computation phase.

- **A low degree curve through $\boldsymbol{\alpha}_i$:** Once we have the evaluation of f on all points in $\mathbb{F}_{p^b}^n$, we initiate some *local* computation at each $\boldsymbol{\alpha}_i$. This local computation would run in time $(adn)^c$ for some fixed constant c, thereby giving an upper bound of $\tilde{O}\left((pad)^n + N(adn)^{O(1)}\right)$ on the total running time. To describe this local computation, let us focus on a point $\boldsymbol{\alpha}_i$. Since the field elements of $\mathbb{F}_q$ are represented as univariate polynomials of degree at most $(a-1)$ in $\mathbb{F}_p[Y]$, we get that for every $\boldsymbol{\alpha}_i \in \mathbb{F}_q^n$, there exist vectors $\boldsymbol{\alpha}_{i,0}, \boldsymbol{\alpha}_{i,1}, \ldots, \boldsymbol{\alpha}_{i,a-1}$ in $\mathbb{F}_p^n$ such that

  $$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1} Y + \cdots + \boldsymbol{\alpha}_{i,a-1} Y^{a-1} .$$

  Let us now consider the curve $\mathbf{g}(t) \in \mathbb{F}_p^n[t]$ defined as

  $$\mathbf{g}_i(t) = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1} t + \cdots + \boldsymbol{\alpha}_{i,a-1} t^{a-1} .$$

  We are interested in some simple properties of this curve. The first such property is that it passes through the point $\boldsymbol{\alpha}_i$, since $\boldsymbol{\alpha}_i = \mathbf{g}_i(Y)$ (recall that Y is an element of $\mathbb{F}_q = \mathbb{F}_p[Y]/\langle v(Y) \rangle$ here). The second property is that this curve contains *a lot* of points in the $\mathbb{F}_{p^b}^n$. In particular, note that for every $\gamma \in \mathbb{F}_{p^b}$, $\mathbf{g}_i(\gamma) \in \mathbb{F}_{p^b}^n$. Thus, there are at least p^b points on $\mathbf{g}_i(t)$ in $\mathbb{F}_{p^b}^n$ (counted with multiplicities).

- **Restriction of f to $\mathbf{g}_i(t)$:** We now look at the univariate polynomial $h_i(t)$ obtained by restricting the n-variate polynomial f to the curve $\mathbf{g}_i(t)$. Thus, if $\mathbf{g}_i(t) = (g_{i,0}(t), \ldots, g_{i,n-1}(t))$ for some univariate polynomials $g_{i,j}(t)$ of degree at most $a-1$, then $h_i(t)$ is equal to the polynomial $f(g_{i,0}(t), \ldots, g_{i,n-1}(t))$. Clearly, the degree of h_i is at most $a(d-1)n < adn$. From our previous discussion, we know that $h_i(Y) = f(\boldsymbol{\alpha}_i)$. Moreover, we have already evaluated f on all of $\mathbb{F}_{p^b}^n$ and thus, we know the value of $h_i(\gamma)$ for all $\gamma \in \mathbb{F}_{p^b}$. Note that these are at least p^b many inputs on which the value of $h_i(t)$ is correctly known to us. Also, from our choice of b, we know that $p^b > adn > \deg(h_i)$. Thus, we can recover the polynomial h_i completely using univariate polynomial interpolation in time at most $\text{poly}(a, d, n, p)$, and thus can output $h_i(Y) = f(\boldsymbol{\alpha}_i)$ in time $\text{poly}(a, d, n, p)$. Iterating this local computation for every $i \in \{0, 1, \ldots, N-1\}$, we can compute the value of f at $\boldsymbol{\alpha}_i$ for each such i.

Correctness and running time. The correctness of the algorithm immediately follows from the outline above. Essentially, we set things up in a way that to compute $f(\boldsymbol{\alpha}_i)$ it suffices to evaluate the univariate polynomial h_i at input $Y \in \mathbb{F}_q$. Moreover, from the preprocessing phase, we already

have the value of f on $\mathbb{F}_{p^b}^n$ and this in turn gives us the evaluation of $h_i(t)$ on $p^b > adn > \deg(h_i)$ distinct inputs. Thus, by standard univariate polynomial interpolation, we recover h_i and hence $h_i(Y) = f(\boldsymbol{\alpha}_i)$ correctly.

The time complexity of the preprocessing phase is dominated by the step where we evaluate f on $\mathbb{F}_{p^b}^n$. This can be upper bounded by $\tilde{O}((padn)^n)$ using the standard multidimensional FFT algorithm. In the local computation phase, the computation at each input point $\boldsymbol{\alpha}_i$ involves constructing the curve $\mathbf{g}_i(t)$, constructing the set $\{(\gamma, h_i(\gamma)) : \gamma \in \mathbb{F}_{p^b}\}$, using the evaluation of h_i on these p^b inputs to recover h_i uniquely via interpolation and then computing $h_i(Y)$. For every $\gamma \in \mathbb{F}_{p^b}$, $\mathbf{g}_i(\gamma) \in \mathbb{F}_{p^b}^n$ can be done in time at most $\mathrm{poly}(a, d, n, p)$. So, the total time complexity of this phase is at most $(N \cdot \mathrm{poly}(a, d, n, p))$, and hence the total running time of the algorithm is $\tilde{O}(N + (padn)^n)$.

3.2 Towards faster multipoint evaluation

The algorithm outlined in the previous section achieves a $O(Nn + d^n)^{1+o(1)}$ when $apn = d^{o(1)}$. We now try to modify it so that it continues to be nearly linear time even when the number of variables n and the degree of underlying field a are not less than $d^{o(1)}$. The factor of p^n appears to be inherent to our approach and seems difficult to get rid of, and this leads to the restriction of working over fields of small characteristic for all our results in this paper.

Before proceeding further, we remark that the basic intuition underlying all of our subsequent algorithms are essentially the same as those in the simple algorithm outlined in this section. For each of the further improvements, we modify certain aspects of this algorithm using a few more technical (and yet simple) ideas on top of the ones already discussed in subsection 3.1.

Handling large number of variables. The factor of n^n in the running time appears in the preprocessing phase of the algorithm in subsection 3.1. The necessity for this stems from the fact that the univariate polynomial $h_i(t)$ obtained by restricting f to the curve $\mathbf{g}_i(t)$ through $\boldsymbol{\alpha}_i$ can have degree as large as $a(d-1)n$. Thus, for interpolating $h_i(t)$ from its evaluations, we need its value on at least $a(d-1)n+1$ distinct inputs. Thus, we need p^b to be at least $a(d-1)n+1$.

However, we note that if we have access to not just the evaluations of $h_i(t)$, but also to the evaluations of its derivatives up to order $n-1$ at each of these inputs in $\mathbb{F}_{p^b}$, then $h_i(t)$ can be uniquely from this information provided p^b is at least $\deg(h_i(t))/n$, i.e. $(a(d-1)n+1)/n \leq ad$ (see Lemma 4.10 for a formal statement). Thus, with observation at hand, we now choose b such that $p^{b-1} \leq ad \leq p^b$. Moreover, for the local computation, we now need not only the evaluation of h_i on all points in $\mathbb{F}_{p^b}$ but also the evaluations of all derivatives of $h_i(t)$ of order at most $n-1$ on all these points. A natural way of ensuring that the evaluations of these derivatives of $h_i(t)$ are available in the local computation phase is to compute not just the evaluation of f but also of *all* its partial derivatives of up to n on all of $\mathbb{F}_{p^b}^n$. Together with the chain rule of partial derivatives, we

can use the evaluations of these partial derivatives of f and the identity $h_i(t) = f \circ \mathbf{g}_i(t)$ to obtain the evaluations of $h_i(t)$ and all its derivatives of order at most $n-1$ on all inputs in $\mathbb{F}_{p^b}$. This ensures that h_i can once again be correctly and uniquely recovered given this information via a standard instance of Hermite Interpolation, which in turn ensures the correctness of the algorithm.

To see the effect on the running time, note that in the preprocessing phase, we now need to evaluate not just f but all its partial derivatives of order at most $n-1$ on all of $\mathbb{F}_{p^b}^n$. Thus, there are now roughly $\binom{n+n}{n} \leq 4^n$ polynomials to work with in this phase. So, given the coefficients of f, we first obtain the coefficients of all these derivatives, and then evaluate these polynomials on $\mathbb{F}_{p^b}^n$ using a multidimensional FFT algorithm again. Also, the coefficient representation of any fixed derivative of order up to $n-1$ can be computed from the coefficients of f in $\tilde{O}(d^n)$ time (see Lemma 4.7). Thus, the total time complexity of the preprocessing phase in this new algorithm can be upper bounded by $\tilde{O}((adp)^n 4^n)$.

Once we have this stronger guarantee from the preprocessing phase, we get to doing some local computation at each point $\boldsymbol{\alpha}_i$. Now, instead of recovering h_i via a standard univariate polynomial interpolation, we have to rely on a standard Hermite interpolation for this. In particular, we need access to the evaluation of all derivatives of $h_i(t)$ of order at most $n-1$ on all inputs $\gamma \in \mathbb{F}_{p^b}$. This can be done via an application of chain rule of derivatives and the fact that we have evaluations of *all* partial derivatives of f of order at most $n-1$ on all points in $\mathbb{F}_{p^b}^n$. The time taken for this computation at each $\gamma \in \mathbb{F}_{p^b}$ turns out to be about $O(4^n \operatorname{poly}(d, n, a, p))$. Thus, the total time taken for local computation at all the input points can be upper bounded by roughly $O(N 4^n \operatorname{poly}(d, n, a, p))$.

Thus, the total time complexity of this modified algorithm is $\tilde{O}((N + (adp)^n)4^n)$. In other words, we have managed to remove the factor of n^n present in the algorithm in subsection 3.1 and replace it by 4^n. An algorithm based on this improvement is described in section 6.

Handling larger fields. We now discuss the improvement in the dependence on the parameter a, which is the degree of the extension of $\mathbb{F}_p$ where the input points lie. In the local computation step at each point, the curve $\mathbf{g}_i(t)$ through $\boldsymbol{\alpha}_i$ has degree $a-1$ in the worst case, since we view the field elements in $\mathbb{F}_{p^a}$ as univariate polynomials of degree at most $a-1$ with coefficients in $\mathbb{F}_p$. Therefore, the restriction of f to such a curve, namely the polynomial $h_i(t)$ can have degree $(a-1)\deg(f)$ in the worst case. This forces us to choose the parameter b such that p^b is at least $\deg(h_i)$, thereby leading to a factor of a^n in the running time. Note that if we had the additional promise that the point $\boldsymbol{\alpha}_i$ was in an extension $\mathbb{F}_{p^{\tilde{a}}}$ of $\mathbb{F}_p$ for some $\tilde{a} < a$, then the curve $\mathbf{g}_i$ would be of degree at most $(\tilde{a}-1) < (a-1)$ and hence the polynomial h_i would have degree at most $(\tilde{a}-1)\deg(f)$. More generally, if all the input points $\boldsymbol{\alpha}_i$ were promised to be in $\mathbb{F}_{p^{\tilde{a}}}^n$, we can improve the factor a^n to $(\tilde{a})^n$ in the running time by choosing b such that p^b is larger than $\tilde{a}dn$ (in fact, we only need $p^b \geq (\tilde{a}d)$ if we are working with multiplicities). We also note that for every

$\tilde{a} \in \mathbb{N}$ the curve $\mathbf{g}_i(t)$ takes a value in $\mathbb{F}_{p^{\tilde{a}}}^n$ whenever t is set to a value in $\mathbb{F}_{p^{\tilde{a}}}$. As a consequence, the curve $\mathbf{g}_i$ contains at least $p^{\tilde{a}}$ points in $\mathbb{F}_{p^{\tilde{a}}}^n$. With these observations in hand, we now elaborate on the idea for reducing the a^n factor in the running time. For simplicity of exposition, we outline our ideas in the setting of the algorithm discussed in subsection 3.1. In particular, derivative based improvements are not involved.

Let a' be such that $p^{a'} > adn \geq p^{a'-1}$. Now, instead of recovering h_i directly from its values on $\mathbb{F}_{p^b}$, we try to recover h_i in two steps. In the first step, we try to obtain the values of $h_i(\gamma)$ for every $\gamma \in \mathbb{F}_{p^{a'}}$ using the information we have from the preprocessing phase. Assuming that we can do this, we can again obtain h_i by interpolation and compute $h_i(Y) = f(\boldsymbol{\alpha}_i)$.

Now, to compute $h_i(\gamma)$ for $\gamma \in \mathbb{F}_{p^{a'}}$, we note that $h_i(\gamma)$ equals $f \circ \mathbf{g}_i(\gamma)$, thus it would be sufficient if we had the evaluation of f on the point set $\{\mathbf{g}_i(\gamma) : \gamma \in \mathbb{F}_{p^{a'}}\}$. This seems like the problem we had started with, but with one key difference: the points $\{\mathbf{g}_i(\gamma) : \gamma \in \mathbb{F}_{p^{a'}}\}$ are all in $\mathbb{F}_{p^{a'}}^n$ with $a' = \Theta(\log adn)$! Thus, the degree of the extension where these points lie is significantly reduced. In essence, this discussion gives us a reduction from the problem of evaluating f on N points in $\mathbb{F}_{p^a}^n$ to evaluating f on $N \cdot adn$ points in $\mathbb{F}_{p^{a'}}^n$, with $a' = \Theta(\log adn)$. Thus, we have another instance of multipoint evaluation with a multiplicatively larger point set in an extension of $\mathbb{F}_p$ of degree logarithmic in adn. If we now apply the algorithm discussed in subsection 3.1, we get a running time of roughly $\tilde{O}(Nadn + (pdn \log(adn))^n)$. Thus, in the running time, the factor a^n has been replaced by $\log^n a$ at the cost of N being replaced by $Nadn$. In fact, we can continue this process ℓ times, and in each step we end up with an instance of multipoint evaluation with the size of the point set being increased by a multiplicative factor, with the gain being that we have a substantial reduction in the degree of the field extension that the points live in.

This idea can be combined with those used for improving the dependence on the number of variables, to get our final algorithm that achieves nearly linear running time provided that $p = d^{o(1)}$ and $a \leq \exp(\exp(\ldots(\exp(d))))$ where the height of this tower of exponentials is fixed. We refer to Theorem 7.1 for a formal statement of the result and section 7 for further details.

Comparison with the techniques of Björklund, Kaski and Williams [BKW19]. Now that we have an overview of the algorithms for multipoint evaluation in this paper, we can elaborate on the similarities they share with the algorithms in [BKW19]. At a high level, the similarities are significant. In particular, both the algorithms have a preprocessing phase where the polynomial is on a product set using multidimensional FFT. This is followed by a local computation step, where the value of the polynomial at any specific input of interest is deduced from the already computed data by working with the restriction of the multivariate polynomial to an appropriate curve. In spite of these similarities in the high level outline, the quantitative details of these algorithms are different. One salient difference is that the time complexity of the algorithm in [BKW19], depends polynomially on the size of the underlying field, whereas in our algorithm outlined above, this

dependence is polynomial in logarithm of the field size as long as the size of the field is bounded by a tower function of fixed height in the degree parameter d. This difference stems from technical differences in the precise product set used in the preprocessing phase and the sets of curves utilized in the local computation phase. In particular, the degree of the curves in the local computation phase of our algorithms depends polynomially on $\log |\mathbb{F}|$, where as the degree of the curves used in [BKW19] depends polynomially on $|\mathbb{F}|$. Additionally, algorithms in [BKW19] rely on the assumption that the total degree of the polynomial divides $|\mathbb{F}^*| - 1$, whereas we do not need any such divisibility condition.

3.3 Data structure for polynomial evaluation

The multipoint evaluation algorithm in Theorem 2.1 is naturally conducive to obtaining data structures for polynomial evaluation. Essentially, the evaluation of the polynomial in a fixed grid (independent of the N points of interest in the input) gives us the data structure, and the local computation at each input point of interest which requires access to some of the information computed in the preprocessing phase constitutes the query phase of the data structure. We discuss this in some more detail now.

Let $f(X) \in \mathbb{F}_{p^a}[X]$ be a univariate polynomial of degree at most n. We start by picking parameters d, m such that d^m is at least n. For any such choice of d and n, there is clearly an m-variate polynomial $F(Z_0, Z_1, \ldots, Z_{m-1})$ such that $F(X, X^d, X^{d^2}, \ldots, X^{d^{m-1}}) = f(X)$. In other words, the image of F under the Kronecker substitution equals f. Now, as in the multipoint evaluation algorithms, we pick the smallest integer b such that $p^b > adm$ and evaluate F on $\mathbb{F}_{p^b}^m$ and store these points along with the value of F on these inputs in the memory. This forms the memory content of our data structure. Thus, the memory can be thought of having $p^{bm} \leq (padm)^m$ cells, each containing a pair $(\mathbf{c}, F(\mathbf{c}))$ for $\mathbf{c} \in \mathbb{F}_{p^b}^m$.

Let us now consider the query complexity of this data structure. Let $\alpha \in \mathbb{F}_{p^a}$ be an input and the goal is to compute $f(\alpha)$. From the relation between F and f, we have that $f(\alpha) = F(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha} = (\alpha, \alpha^d, \alpha^{d^2}, \ldots, \alpha^{d^{m-1}})$. Now, we rely on the local computation in the multipoint evaluation algorithms to compute $F(\boldsymbol{\alpha})$. In the algorithm, we consider a curve $\mathbf{g}$ of degree at most $a - 1$ which passes through $\boldsymbol{\alpha}$ and look at the restriction of F to this curve to get a univariate polynomial h of degree less than adm. Then, we take the value of h on inputs in $\mathbb{F}_{p^b}$, which can be recovered from the value of F on the points in the set $\mathbf{g}(\mathbb{F}_{p^b}) \cap \mathbb{F}_{p^b}^n$. Finally, note that there are at least $p^b > adm$ of these inputs and value of h on these inputs is already stored in the memory. This suffices to recover h and thus, also $f(\alpha) = F(\boldsymbol{\alpha})$. So, the query complexity of this data structure is adm.

To get a sense of the parameters, let us set $d = n^{1/\log\log n}$ and $m = \log\log n$. Clearly, the constraint $d^m \geq n$ is met in this case. For this choice of parameter and for p being a constant and $a \leq \mathrm{poly}(\log n)$, we get that the space complexity is at most $n^{1+o(1)}$ and the query complexity is at most $n^{o(1)}$.

The complete details can be found in section 8.

3.4 Rigidity of Vandermonde matrices

The connection between rigidity of Vandermonde matrices and multipoint evaluation is also quite natural. Consider a Vandermonde matrix V_n with generators $\alpha_0, \ldots, \alpha_{n-1}$ and for every $i, j \in \{0, 1, \ldots, n-1\}$, the (i, j)th entry of V_n is α_i^j. Now, for any univariate polynomial f of degree at most $n-1$, the coefficients of f, together with the set $\{\alpha_i : i \in \{0, 1, \ldots, n-1\}\}$ of generators form an instance of (univariate) multipoint evaluation. Moreover, for any choice of the generators $\{\alpha_i : i \in \{0, 1, \ldots, n-1\}\}$, the algorithm for multipoint evaluation, e.g Theorem 2.1 can naturally be interpreted as a circuit for computing the linear transform given by the matrix V_n. Furthermore, if this linear circuit is structured enough, we could, in principle hope to get a decomposition of V_n as a sum of a sparse and a low rank matrix from this linear circuit, for instance, along the lines of the combinatorial argument of Valiant [Val77]. Our proof of Theorem 2.6 is along this outline. We now describe these ideas in a bit more detail.

Given a univariate polynomial f of degree $n-1$ and inputs $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, let F be an m-variate polynomial of degree d such that $(n = d^m)$[9] as described in subsection 3.3. Moreover, for $i \in \{0, 1, \ldots, n-1\}$, let $\boldsymbol{\alpha}_i = (\alpha_i, \alpha_i^d, \ldots, \alpha_i^{d^{m-1}})$. Now, as discussed in subsection 3.3, $f(\alpha_i) = F(\boldsymbol{\alpha}_i)$. Let $\tilde{V}$ be the $n \times n$ matrix where the rows are indexed by $\{0, 1, \ldots, n-1\}$ and the columns are indexed by all m- variate monomials of individual degree at most $d-1$. We use the fact that $d^m = n$ here. From the above set up, it immediately follows that the coefficient vectors of f and F are equal to each other (with the coordinate indices having slightly different semantics) and the matrices V_n and $\tilde{V}$ are equal to each other.

We now observe that the algorithm for multipoint evaluation described in subsection 3.1 gives a natural decomposition of $\tilde{V}$ (and hence V_n) as a product of a matrix A of row sparsity at most adm and a $p^{bm} \times d^m$ matrix B with b being the smallest integer such that $p^b > adm$. The rows of B are indexed by all elements of $\mathbb{F}_{p^b}^m$ and the columns are indexed by all m-variate monomials of individual degree at most $d-1$, and the $(\boldsymbol{\alpha}, \mathbf{e})$ entry of B equals $\boldsymbol{\alpha}^{\mathbf{e}}$. Intuitively, the matrix B corresponds to the preprocessing phase of the algorithm and the matrix A corresponds to the local computation. At this point, we use an upper bound of [DL20] on the rigidity of Discrete Fourier Transform matrices over finite fields and the inherent Kronecker product structure of the matrix B to obtain an upper bound on the rigidity of B. Finally, we observe that that matrix $V_n = \tilde{V} = A \cdot B$ obtained by multiplying a sufficiently non-rigid matrix B with a row sparse matrix A continues to be non-rigid with an interesting regime of parameters. This essentially completes the proof. For more details, we refer the reader to section 9.

[9] For simplicity, let us assume that such a choice of integers d, m exist.

4 Preliminaries

We use $\mathbb{N}$ to denote the set of natural numbers $\{0, 1, 2, \ldots\}$, $\mathbb{F}$ to denote a general field. For any positive integer N, $[N]$ denotes the set $\{1, 2, \ldots, N\}$. By $\mathbf{x}$ and $\mathbf{z}$, we denote the variable tuples $(X_1, \ldots, X_n)$ and $(Z_1, \ldots, Z_n)$, respectively. For any $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$, $\mathbf{x}^{\mathbf{e}}$ denotes the monomial $\prod_{i=1}^{n} X_i^{e_i}$. By $|\mathbf{e}|_1$, we denote the sum $e_1 + \cdots + e_n$.

For every positive integer k, $k!$ denotes $\prod_{i=1}^{k} i$. For $k = 0$, $k!$ is defined as 1. For two non-negative integer i and k with $k \geq i$, $\binom{k}{i}$ denotes $\frac{k!}{i!(k-i)!}$. For $k < i$, $\binom{k}{i} = 0$. For non-negative integer $i_1, \ldots, i_s$ with $i_1 + \cdots + i_s = k$, $\binom{k}{i_1, \ldots, i_s} = \frac{k!}{i_1! \cdots i_s!}$. For $\mathbf{a} = (a_1, \ldots, a_n), \mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$, $\binom{\mathbf{a}}{\mathbf{b}} = \prod_{i=1}^{n} \binom{a_i}{b_i}$, and $\binom{\mathbf{a}+\mathbf{b}}{\mathbf{a},\mathbf{b}} = \prod_{i=1}^{n} \binom{a_i+b_i}{a_i,b_i}$.

We say that a function $\psi : \mathbb{N} \to \mathbb{N}$ is polynomially bounded, or denoted by $\psi(n) \leq \mathrm{poly}(n)$, if there exists a constant c such that for all large enough $n \in \mathbb{N}$, $\psi(n) \leq n^c$.

Proposition 4.1. *For any two positive integers i and k with $k \geq i$,*

$$\binom{k}{i} \leq \left(\frac{ke}{i}\right)^i.$$

For proof see [Juk10, Chapter 1]. Suppose that p be a positive integer greater than 1. Then for any non-negative integer c, $\log_p^{\circ c}(n)$ denotes the c-times composition of logarithm function with itself, with respect to base p. For example, $\log_p^{\circ 2}(n) = \log_p \log_p(n)$. By $\log_p^{\star}(n)$, denotes the smallest non-negative integer c such that $\log_p^{\circ c}(n) \leq 1$. For $p = 2$, we may omit the subscript p in $\log_p(n)$, $\log_p^{\circ c}(n)$ and $\log_p^{\star}(n)$.

4.1 Some facts about finite fields

Suppose that p is a prime and $q = p^a$ for some positive integer a. Then there exists an *unique* finite field of size q. In other words, all the finite fields of size q are *isomorphic* to each other. We use $\mathbb{F}_q$ to denote the finite field of size q, and p is called the characteristic of $\mathbb{F}_q$. For any finite field $\mathbb{F}_q$, $\mathbb{F}_q^*$ represents the multiplicative cyclic group after discarding the field element 0. For any irreducible polynomial $v(Y)$ over $\mathbb{F}_q$, the quotient ring $\mathbb{F}_q[Y]/\langle v(Y) \rangle$ forms a larger field over $\mathbb{F}_q$ of size q^b where b is the degree of $v(Y)$. The next lemma describes that we can efficiently construct such larger fields over $\mathbb{F}_q$, when the characteristic of the field is small.

Lemma 4.2. *Let p be a prime and $q = p^a$ for some positive integer a. Then, for any positive integer b, the field $\mathbb{F}_{q^b}$ can be constructed as $\mathbb{F}_q[Y]/\langle v(Y) \rangle$, where $v(Y)$ is degree b irreducible polynomial over $\mathbb{F}_q$, in $\mathrm{poly}(a, b, p)$ $\mathbb{F}_q$-operations. Furthermore, all the basic operations in $\mathbb{F}_{q^b}$ can be done in $\mathrm{poly}(b)$ $\mathbb{F}_q$-operations.*

Proof. The elements of the quotient ring $\mathbb{F}_q[Y]/\langle v(Y) \rangle$ are polynomials in Y over $\mathbb{F}_q$ with degree less than b, and the operations are polynomial addition and multiplication under modulo $v(Y)$.

Therefore, once we have an irreducible $v(Y)$ (over $\mathbb{F}_q$) of degree b, we can perform the basic operations in $\mathbb{F}_{q^b}$ using $\text{poly}(b)$ $\mathbb{F}_q$-operations. From [Sho90, Theorem 4.1], we can compute a degree b irreducible polynomial $v(Y)$ over $\mathbb{F}_q$ using $\text{poly}(a, b, p)$ $\mathbb{F}_p$-operations. □

Fix a field F_q of characteristic p. In the standard algebraic model over $\mathbb{F}_q$, the basic operations are addition, subtraction, multiplication, and division of elements in $\mathbb{F}_q$. Let $\mathbb{F}_q = \mathbb{F}_p[X]/\langle g(X)\rangle$ where $q = p^a$ and $g(X)$ is a degree a irreducible polynomial over $\mathbb{F}_p$. Then for any element $\alpha \in \mathbb{F}_q$, consider its canonical representation $\alpha = \alpha_0 + \alpha_1 X + \ldots + \alpha_{a-1}X^{a-1}$ where $\alpha_i \in \mathbb{F}_p$. Note that it is not clear how to extract α_i's from α using the algebraic operations over $\mathbb{F}_q$. We show that this is possible if p is small. Since $\mathbb{F}_q = \mathbb{F}_p[X]/\langle g(X)\rangle$, $X \in \mathbb{F}_q$ is a root of the degree a irreducible polynomial $g(X)$ (over $\mathbb{F}_p$). This implies that $X, X^p, X^{p^2}, \ldots, X^{p^{a-1}}$ are all distinct elements of $\mathbb{F}_q$.

Lemma 4.3. *Let p be prime and $q = p^a$ for some positive integer a. Let $\mathbb{F}_q = \mathbb{F}_p[X]/\langle g(X)\rangle$ where $g(X)$ is a degree a irreducible polynomial over $\mathbb{F}_p$. Let $\alpha \in \mathbb{F}_q$ and $\alpha = \alpha_0 + \alpha_1 X + \cdots + \alpha_{a-1}X^{a-1}$ where $\alpha_i \in \mathbb{F}_p$. Then, given blackbox access to α and $\mathbb{F}_q$-operations, $\alpha_0, \alpha_1, \ldots, \alpha_{a-1}$ can be computed in $\text{poly}(a, \log p)$ $\mathbb{F}_q$-operations.*

Proof. Note that, given α, we can compute α^p by repeated squaring over $\mathbb{F}_q$. Applying this iteratively, we have access to all conjugates $\alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{a-1}}$. Observe that,

$$\underbrace{\begin{bmatrix} 1 & X & X^2 & \ldots & X^{a-1} \\ 1 & X^p & X^{2p} & \ldots & X^{p(a-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & X^{p^{a-1}} & X^{2p^{a-1}} & \ldots & X^{(a-1)p^{a-1}} \end{bmatrix}}_{A} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \ldots \\ \alpha_{a-1} \end{bmatrix} = \begin{bmatrix} \alpha \\ \alpha^p \\ \ldots \\ \alpha^{p^{a-1}} \end{bmatrix}.$$

Note that, the matrix A in the above linear system is a Vandermonde matrix and thus invertible. Also, each entry of A is an element in $\mathbb{F}_q$. Thus, we can find α_i by solving this linear system over $\mathbb{F}_q$. For time complexity, note that we can use α^{p^i} to compute $\alpha^{p^{i+1}}$. Thus, $\alpha, \alpha^p, \ldots, \alpha^{p^{a-1}}$ can be computed in $\text{poly}(a, \log p)$ $\mathbb{F}_q$-operations. Also, the computation of A and solving the linear system can be done in $\text{poly}(a, \log p)$ $\mathbb{F}_q$-operations. Therefore, overall complexity is $\text{poly}(a, \log p)$ $\mathbb{F}_q$-operations. □

Thus, for the rest of our paper, we consider that the extraction of the $\mathbb{F}_p$-coefficients from elements in $\mathbb{F}_q$ as an algebraic operation. Also, in our applications, the time complexity overhead introduced due to this is negligible.

Suppose that $\mathbb{F}_{q_1}$ and $\mathbb{F}_{q_2}$ are two finite fields of characteristic p such that $\mathbb{F}_{q_1}$ is a subfield of $\mathbb{F}_{q_2}$. Then $\mathbb{F}_{q_2}$ forms a vector space over $\mathbb{F}_{q_1}$. A subset $\{\beta_1, \beta_2, \ldots, \beta_k\}$ of $\mathbb{F}_{q_2}$ is called an $\mathbb{F}_{q_1}$-basis if every element of $\alpha \in F_{q_2}$ is a unique linear combination of β_i's over $\mathbb{F}_{q_1}$.

Lemma 4.4. *Let p be a prime and $q = p^a$ for some positive integer a. Let b be a positive integer and $\mathbb{F}_{q^b} = \mathbb{F}_q[Y]/\langle v(Y)\rangle$ for some degree b irreducible polynomial $v(Y)$ over $\mathbb{F}_q$. Then, the following holds:*

1. *The field $\mathbb{F}_{q^b}$ contains the subfield $\mathbb{F}_{p^b}$. Furthermore, all the elements of $\mathbb{F}_{p^b}$ can be computed in $p^b \cdot \mathrm{poly}(a, b, p)$ $\mathbb{F}_q$-operations.*
2. *In $\mathrm{poly}(a, b, p)$ $\mathbb{F}_q$-operations, an element $\beta \in \mathbb{F}_{q^b}$ can be computed such that $\{1, \beta, \ldots, \beta^{b-1}\}$ forms an $\mathbb{F}_p$-basis for $\mathbb{F}_{p^b}$. Moreover, given any element $\alpha \in \mathbb{F}_{p^b}$, the $\mathbb{F}_p$-linear combination of α in the basis $\{1, \beta, \ldots, \beta^{b-1}\}$ can be computed in $\mathrm{poly}(b)$ $\mathbb{F}_q$-operations.*

Proof. Since $p^b - 1$ divides $q^b - 1$, $\mathbb{F}_{q^b}$ is a splitting field of the $x^{p^b} - x$, that is, $x^{p^b} - x$ linearly factorizes over $\mathbb{F}_{q^b}$. Now, one can show that the roots of $x^{p^b} - x$ over $\mathbb{F}_{q^b}$ form a subfield of size p^b. Now, using [Sho90, Theorem 3.2], we can compute a degree b irreducible polynomial $u(Z)$ over $\mathbb{F}_p$ in $\mathrm{poly}(b, p)$ $\mathbb{F}_p$-operations. Next, applying [Ber70], we can find a root $\beta \in \mathbb{F}_{q^b}$ for $u(Z)$ in $\mathrm{poly}(a, b, p)$ $\mathbb{F}_q$-operations. One can show that for any other polynomial $u'(Z)$ with $u'(\beta) = 0$, $u(Z)$ divides $u'(Z)$. Also, β is in $\mathbb{F}_{p^b}$ since $\mathbb{F}_{p^b}$ is a splitting field for $u(Z)$. This implies that $\{1, \beta, \ldots, \beta^{b-1}\}$ forms an $\mathbb{F}_p$-basis for $\mathbb{F}_{p^b}$. Thus, after having β, we can compute all the elements of $\mathbb{F}_{p^b}$ by taking all possible $\mathbb{F}_p$-linear combinations of $\{1, \beta, \ldots, \beta^{b-1}\}$. The cost of doing this is $p^b \cdot \mathrm{poly}(b, p)$ $\mathbb{F}_q$-operations. Computing β takes $\mathrm{poly}(a, b, p)$ $\mathbb{F}_q$-operations. Therefore, in $p^b \cdot \mathrm{poly}(a, b, p)$ $\mathbb{F}_q$-operations, we can compute all the elements of $\mathbb{F}_{p^b}$.

Let $\alpha \in \mathbb{F}_{p^b}$. Since $\mathbb{F}_{p^b}$ is a subfield of $\mathbb{F}_{q^b}$, from the representation of $\mathbb{F}_{q^b}$, we can write $\alpha = \alpha_0 + \alpha_1 Y + \cdots + \alpha_{b-1} Y^{b-1}$ where $\alpha_i \in \mathbb{F}_q$. Also, for all $i \in \{0, 1, \ldots, b-1\}$, each β^i can be written as $\beta_{i,0} + \beta_{i,1} Y + \cdots + \beta_{i,b-1} Y^{b-1}$ where $\beta_{i,j} \in \mathbb{F}_q$. Let $\alpha = c_0 + c_1\beta + \cdots + c_{b-1}\beta^{b-1}$, where c_i's are unknown and we want to find them. This combined with the representation of α and β^i, we get a system of linear equations in $\{c_0, \ldots, c_{b-1}\}$ over $\mathbb{F}_q$. Now we can solve it in $\mathrm{poly}(b)$ $\mathbb{F}_q$-operations and get c_i's. □

4.2 Hasse derivatives

In this section, we briefly discuss the notion of Hasse derivatives that plays a crucial role in our results.

Definition 4.5 (Hasse derivative)**.** *Let $f(\mathbf{x})$ be an n-variate polynomial over a field $\mathbb{F}$. Let $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$. Then, the Hasse derivative of f with respect to the monomial $\mathbf{x}^\mathbf{e}$ is the coefficient of $\mathbf{z}^\mathbf{e}$ in the polynomial $f(\mathbf{x} + \mathbf{z}) \in (\mathbb{F}[\mathbf{x}])[\mathbf{z}]$.* ⌟

Notations. Suppose that $f(\mathbf{x})$ be an n-variate polynomial over a field $\mathbb{F}$. Let $\mathbf{b} \in \mathbb{N}^n$. Then, $\overline{\partial}_\mathbf{b}(f)$ denotes the Hasse derivative of $f(\mathbf{x})$ with respect to the monomial $\mathbf{x}^\mathbf{b}$. For any non-negative integer k, $\overline{\partial}^{\leq k}(f)$ is defined as

$$\overline{\partial}^{\leq k}(f) = \left\{ \overline{\partial}_\mathbf{b}(f) \mid \mathbf{b} \in \mathbb{N}^n \text{ s.t. } |\mathbf{b}|_1 \leq k \right\},$$

and $\overline{\partial}^{<k}(f)$ denotes the set $\{\overline{\partial}_{\mathbf{b}}(f) \mid \mathbf{b} \in \mathbb{N}^n \text{ s.t. } |\mathbf{b}|_1 < k\}$.

For a univariate polynomial $h(t)$ over $\mathbb{F}$ and a non-negative integer k, $h^{(k)}(t)$ denotes the Hasse derivative of $h(t)$ with respect to the monomial t^k, that is, $\text{Coeff}_{Z^k}(h(t+Z))$.

Next, we mention some useful properties of Hasse derivatives.

Proposition 4.6. *Let $f(\mathbf{x})$ be an n-variate polynomial over $\mathbb{F}$. Let $\mathbf{a}, \mathbf{b} \in \mathbb{N}^n$. Then,*

1. $\overline{\partial}_{\mathbf{a}}(f) = \sum_{\mathbf{e} \in \mathbb{N}^n} \binom{\mathbf{e}}{\mathbf{a}} \text{Coeff}_{\mathbf{x}^{\mathbf{e}}}(f) \mathbf{x}^{\mathbf{e}-\mathbf{a}}$.
2. $\overline{\partial}_{\mathbf{a}} \overline{\partial}_{\mathbf{b}}(f) = \binom{\mathbf{a}+\mathbf{b}}{\mathbf{a},\mathbf{b}} \overline{\partial}_{\mathbf{a}+\mathbf{b}}(f)$.

For proof one can see [For14, Appendix C]. The following lemma describes the cost of computing Hasse derivatives.

Lemma 4.7. *Let p be a prime and $q = p^a$ for some positive integer a. Let $f(\mathbf{x})$ be an n-variate polynomial over $\mathbb{F}_q$ with individual degree less than d. Let $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$. Then, given $f(\mathbf{x})$ and $\mathbf{b}$ as input, Algorithm 1 outputs $\overline{\partial}_{\mathbf{b}}(f)$ in*

$$d^n \cdot \text{poly}(n) + \text{poly}(b, d)$$

$\mathbb{F}_q$-operations, where $b = \max_{i \in [n]} b_i$.

Proof. We first describe the algorithm and then argue about its correctness and running time.

Algorithm 1 Computing Hasse derivative

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d and $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$.
Output: $\bar{\partial}_{\mathbf{b}}(f)$.

1: Let b be $\max_{i \in [n]} b_i$.
2: Let D be an $(b+1) \times d$ array.
3: **for** $j \leftarrow 0$ to $d-1$ **do**
4: **for** $i \leftarrow 0$ to b **do**
5: **if** $i = j$ **then**
6: $D_{i,j} \leftarrow 1$.
7: **else if** $i > j$ **then**
8: $D_{i,j} \leftarrow 0$.
9: **else if** $i = 0$ **then**
10: $D_{0,j} \leftarrow 1$.
11: **else**
12: $D_{i,j} = D_{i-1,j-1} + D_{i,j-1}$
13: **for** $\mathbf{e} \in \{0, 1, \ldots, d-1\}^n$ **do**
14: Let $\mathbf{e} = (e_1, \ldots, e_n)$.
15: $c_{\mathbf{e}} \leftarrow \mathrm{Coeff}_{\mathbf{x}^{\mathbf{e}}}(f) \cdot \prod_{i=1}^{n} D_{b_i, e_i}$.
16: Output $\sum_{\mathbf{e} \in \{0,1,\ldots,d-1\}^n} c_{\mathbf{e}} \mathbf{x}^{\mathbf{e}-\mathbf{b}}$.

In Algorithm 1, for all $i \in \{0, 1, \ldots, b\}$ and $j \in \{0, 1, \ldots, d-1\}$, the (i,j)th entry of array D

$$D_{i,j} = \binom{j}{i} \bmod p.$$

For this, we note that the arithmetic in Line 15 of the algorithm is happening over the underlying field $\mathbb{F}_q$. This combined with Proposition 4.6 implies that the Algorithm 1 computes $\bar{\partial}_{\mathbf{b}}(f)$.

To compute the array D, we are performing $d(b+1)$ $\mathbb{F}_p$-operations. Computing all $c_{\mathbf{e}}$'s for $\mathbf{e} \in \{0, 1, \ldots, d-1\}^n$ takes $d^n \cdot (n+1)$ $\mathbb{F}_q$-operations. Therefore, Algorithm 1 runs in our desired time complexity. □

4.3 Univariate polynomial evaluation and interpolation

The two simplest but most important ways of representing an univariate polynomial of degree less than d are either by giving the list of its coefficients, or by giving its evaluations at d distinct points. In this section, we discuss about the cost of changing between these two representations.

First, we mention the cost of polynomial evaluation, that is, going from the list of coefficients to the list of evaluations.

Lemma 4.8 (Evaluation)**.** *Let $f(x)$ be a degree d polynomial over $\mathbb{F}$. Let $\alpha_1, \alpha_2, \ldots, \alpha_N$ be N distinct elements from $\mathbb{F}$. Then, $f(\alpha_i)$ for all $i \in [N]$ can be computed in $O(Nd)$ $\mathbb{F}$-operations.*

For each $i \in [N]$, using Horner's rule, one can compute $f(\alpha_i)$ with $d-1$ additions and $d-1$ multiplications over $\mathbb{F}$. Therefore, the total cost of computing $f(\alpha_i)$ for all $i \in [N]$ is $O(Nd)$ operations. For more details see [GG03, Section 5.2]. Next, we discuss the cost of polynomial interpolation where we go from the list of evaluations to the list of coefficients.

Lemma 4.9 (Interpolation)**.** *Let $f(x)$ be a degree d polynomial over $\mathbb{F}$. Let $\alpha_0, \alpha_1, \ldots, \alpha_d$ be $(d+1)$ distinct elements from $\mathbb{F}$. Let $\beta_i = f(\alpha_i)$ for all $i \in \{0, 1, \ldots, d\}$. Then, given (α_i, β_i) for all $i \in \{0, 1, \ldots, d\}$, $f(x)$ can be computed in $O(d^2)$ $\mathbb{F}$-operations.*

For proof see [GG03, Section 5.2]. The following lemma gives a stronger version of univariate polynomial interpolation, known as Hermite interpolation. Here, the number of evaluation points can be less than d, but evaluations of Hasse derivatives of the polynomial up to certain order is available.

Lemma 4.10 (Hermite interpolation)**.** *Let $f(x)$ be a degree d univariate polynomial over a field $\mathbb{F}$ and $e_1, \ldots, e_m$ be m positive integers such that $e_1 + \cdots + e_m$ is greater than d. Let $\alpha_1, \ldots, \alpha_m$ be m distinct elements from $\mathbb{F}$. For all $i \in [m]$ and $j \in [e_j]$, let $f^{(j-1)}(\alpha_i) = \beta_{ij}$. Then given $(\alpha_i, j, \beta_{ij})$ for all $i \in [m]$ and $j \in [e_j]$, $f(x)$ can be computed in $O(d^2)$ $\mathbb{F}$-operations.*

For proof see [GG03, Section 5.6]. We also remark that while there are nearly linear time algorithms for all of the above operations (multipoint evaluation, interpolation and Hermite interpolation) based on the Fast Fourier transform; however, for our applications in this paper, the above stated more naive bounds suffice.

4.4 Multidimensional Fast Fourier transform

We crucially rely on the following lemma that says that there is a fast algorithm for evaluating an n-variate polynomial f with coefficients in a finite field $\mathbb{F}$ on the set $\tilde{\mathbb{F}}^n$ where $\tilde{\mathbb{F}}$ is a subfield of $\mathbb{F}$. The proof is based on a simple induction on the number of variables and uses the standard FFT one variable at a time. For the proof, see Theorem 4.1 in [KU11].

Lemma 4.11. *Let $\mathbb{F}$ be a finite field and let $\tilde{\mathbb{F}}$ be a subfield of $\mathbb{F}$. Then, there is a deterministic algorithm that takes as input an n-variate polynomial $f \in \mathbb{F}[\mathbf{x}]$ of degree at most $d-1$ in each variable as a list of coefficients, and in at most $(d^n + |\tilde{\mathbb{F}}|^n) \cdot \mathrm{poly}(n, d, \log|\mathbb{F}|)$ operations over the field $\mathbb{F}$, it outputs the evaluation of f for all $\boldsymbol{\alpha} \in \tilde{\mathbb{F}}^n$.*

5 A simple algorithm for multipoint evaluation

We start with our first and simplest algorithm for multipoint evaluation . The algorithm gives an inferior time complexity to what is claimed in Theorem 2.1, but contains some of the main ideas. Subsequently, in section 6 and section 7, we build upon this algorithm to eventually prove Theorem 2.1. Our main theorem for this section is the following.

Theorem 5.1. *Let p be a prime and $q = p^a$ for some positive integer a. There is a deterministic algorithm such that on input an n-variate polynomial $f(\mathbf{x})$ over $\mathbb{F}_q$ with individual degree less than d and points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$, it outputs $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in time*

$$(N + (adnp)^n) \cdot \mathrm{poly}(a, d, n, p)\,.$$

5.1 A description of the algorithm

We start with a description of the algorithm, followed by its analysis. We recall again that through all the algorithms in this and subsequent sections, we assume that the underlying field $\mathbb{F}_q$ is given to us via an irreducible polynomial of appropriate degree over the prime subfield. Moreover, from Lemma 4.3, we also assume without loss of generality that for every input field element, we have access to its representation as a polynomial of appropriate degree over the prime subfield. For a polynomial map $\mathbf{g}(t) = (g_1(t), g_2(t), \ldots, g_n(t))$ and an n-variate polynomial f, we use $f(\mathbf{g}(t))$ to denote the univariate polynomial $f(g_1(t), g_2(t), \ldots, g_n(t))$.

Algorithm 2 Efficient Multivariate Multipoint Evaluation

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d and N distinct points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$.
Output: $f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \ldots, f(\boldsymbol{\alpha}_N)$.

1: Let p be the characteristic of $\mathbb{F}_q$ and $q = p^a$.
2: Let $v_0(Y_0)$ be an irreducible polynomial in $\mathbb{F}_p[Y_0]$ of degree a and

$$\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle.$$

3: Let b be the smallest integer such that $p^b > adn$.
4: Compute an irreducible polynomial $v_1(Y_1)$ in $\mathbb{F}_q[Y_1]$ of degree b and

$$\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle. \text{ (Lemma 4.2)}$$

5: Compute the subfield $\mathbb{F}_{p^b}$ of $\mathbb{F}_{q^b}$. (Lemma 4.4)
6: Evaluate $f(\mathbf{x})$ over the grid $\mathbb{F}_{p^b}^n$. (Lemma 4.11)
7: **for** all $i \in [N]$ **do**
8: Let $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}Y_0 + \cdots + \boldsymbol{\alpha}_{i,a-1}Y_0^{a-1}$, where $\boldsymbol{\alpha}_{i,j} \in \mathbb{F}_p^n$.
9: Let $\mathbf{g}_i(t)$ be the curve defined as $\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}t + \cdots + \boldsymbol{\alpha}_{i,a-1}t^{a-1}$.
10: Compute the set $P_i = \{(\gamma, \mathbf{g}_i(\gamma)) \mid \gamma \in \mathbb{F}_{p^b}\}$. (Lemma 4.8)
11: Compute the set $E_i = \{(\gamma, f(\boldsymbol{\gamma}')) \mid (\gamma, \boldsymbol{\gamma}') \in P_i\}$ from the evaluations of $f(\mathbf{x})$ over $\mathbb{F}_{p^b}^n$.
12: Let $h_i(t)$ be the univariate polynomial defined as $f(\mathbf{g}_i(t))$.
13: Using E_i, interpolate $h_i(t)$. (Lemma 4.9)
14: Output $h_i(Y_0)$ as $f(\boldsymbol{\alpha}_i)$. (Lemma 4.8)

5.2 Analysis of Algorithm 2

Proof of Theorem 5.1. We start with the proof of correctness of the algorithm.

Correctness of Algorithm 2. We show that Algorithm 2 computes $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in $(N + (adnp)^n) \cdot \mathrm{poly}(a, d, n, p)$ many $\mathbb{F}_q$ operations. We assume that the underlying field $\mathbb{F}_q$ is represented as $\mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$, where $v_0(Y_0)$ is a degree a irreducible polynomial over $\mathbb{F}_p$. From Lemma 4.2, the field $\mathbb{F}_{q^b}$ can be constructed as $\mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$ for some degree b irreducible polynomial $v_1(Y_1)$ over $\mathbb{F}_q$. Lemma 4.4 ensures that we can explicitly compute all the elements of the subfield $\mathbb{F}_{p^b}$ (of $\mathbb{F}_{q^b}$). The representation of $\mathbb{F}_q$ ensures that every element $\beta \in \mathbb{F}_q$ is of the form $\beta_0 + \beta_1 Y_0 + \cdots + \beta_{a-1}Y_0^{a-1}$, where $\beta_i \in \mathbb{F}_p$. Therefore, for all $i \in [N]$, $\boldsymbol{\alpha}_i$ is of the form $\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}Y_0 + \cdots + \boldsymbol{\alpha}_{i,a-1}Y_0^{a-1}$, where $\boldsymbol{\alpha}_{i,j} \in \mathbb{F}_p^n$. For all $i \in [N]$, the curve $\mathbf{g}_i(t)$ is defined as

$\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}t + \cdots + \boldsymbol{\alpha}_{i,a-1}t^{a-1}$. Since f is an n-variate polynomial over $\mathbb{F}_q$ with individual degree less than d, for all $i \in [N]$, the polynomial $h_i(t) = f(\mathbf{g}_i(t))$ is a polynomial in t of degree less than adn. For all $\gamma \in \mathbb{F}_{p^b}$, $\mathbf{g}_i(\gamma)$ is in $\mathbb{F}_{p^b}^n$. Therefore, from the evaluations of $f(\mathbf{x})$ over the grid $\mathbb{F}_{p^b}^n$, we get the set $E_i = \{(\gamma, h_i(\gamma) \mid \gamma \in \mathbb{F}_{p^b}\}$. Since the degree of h_i is less than adn and p^b is greater than adn, from the set E_i, we can interpolate $h_i(t)$. The construction of $\mathbf{g}_i(t)$ ensures that $\mathbf{g}_i(Y_0) = \boldsymbol{\alpha}_i$. Hence, $h_i(Y_0) = f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$.

Time complexity of Algorithm 2. Now we discuss the time complexity of Algorithm 2. From Lemma 4.2, the field $\mathbb{F}_{q^b}$ can be constructed as $\mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$ for some degree b irreducible polynomial $v_1(Y_1)$ over $\mathbb{F}_q$ in $\text{poly}(a, b, p)$ many $\mathbb{F}_q$-operations. Also, all the basic operations in the field $\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$ can be done using $\text{poly}(b)$ $\mathbb{F}_q$-operations. Applying Lemma 4.4, the cost of computing all the elements of the subfield $\mathbb{F}_{p^b}$ (of $\mathbb{F}_{q^b}$) is $p^b \cdot \text{poly}(a, b, p)$ $\mathbb{F}_q$-operations. Using Lemma 4.11, we can evaluate $f(\mathbf{x})$ over the grid $\mathbb{F}_{p^b}^n$ in

$$(d^n + p^{bn}) \cdot \text{poly}(a, b, d, n, p)$$

$\mathbb{F}_q$-operations. For all $i \in [N]$, using Lemma 4.8, the cost of computing the set $P_i = \{(\gamma, \mathbf{g}_i(\gamma)) \mid \gamma \in \mathbb{F}_{p^b}\}$ is $p^b \cdot \text{poly}(a, b, n)$ $\mathbb{F}_q$-operations. Using the set E_i, Lemma 4.9 ensures that $h_i(t)$ can be interpolated using $\text{poly}(a, b, d, n)$ $\mathbb{F}_q$-operations. Finally, $h(Y_0)$ can be computed in $\text{poly}(a, d, n)$ many $\mathbb{F}_q$-operations. Since $adn < p^b \leq adnp$, the above discussion implies that that Algorithm 2 performs

$$(N + (adnp)^n) \cdot \text{poly}(a, d, n, p)$$

$\mathbb{F}_q$-operations. □

6 Multipoint evaluation for large number of variables

In this section, we append the overall structure of Algorithm 2 with some more ideas to improve the dependence of the running time on n. In particular, the goal is to reduce the n^n factor in the running time of Theorem 5.1 to a factor of the form $\exp(O(n))$. The main result of this section is the following theorem.

Theorem 6.1. *Let p be a prime and $q = p^a$ for some positive integer a. There is a deterministic algorithm such that on input an n-variate polynomial $f(\mathbf{x})$ over $\mathbb{F}_q$ with individual degree less than d and points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$, it outputs $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in time*

$$(N + (adp)^n) \cdot 4^n \cdot \text{poly}(a, d, n, p).$$

A useful additional ingredient in the proof of this theorem is the following lemma. Semanti-

cally, this is an explicit form of the chain rule of Hasse derivatives for the restriction of a multivariate polynomial to a curve of low degree.

Lemma 6.2. *Let $f(\mathbf{x})$ be an n-variate degree d polynomial over a field $\mathbb{F}$, $\mathbf{g}(t) = (g_1, \ldots, g_n)$ where $g_i \in \mathbb{F}[t]$, and $h(t) = f(\mathbf{g}(t))$. For all $i \in [n]$, let $g_i(t + Z) = g_i(t) + Z\tilde{g}_i(t, Z)$ for some $\tilde{g}_i \in \mathbb{F}[t, Z]$. Let $\tilde{\mathbf{g}}(t, Z) = (\tilde{g}_1, \ldots, \tilde{g}_n)$, and for all $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$, $\tilde{\mathbf{g}}_{\mathbf{e}} = \prod_{i=1}^{n} \tilde{g}_i^{e_i}$. For any $\ell \in \mathbb{N}$, let*

$$h_\ell(t, Z) = \sum_{i=0}^{\ell} Z^i \sum_{\mathbf{e} \in \mathbb{N}^n : |\mathbf{e}|_1 = i} \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{e}}(t, Z).$$

Then, for every $k \in \mathbb{N}$ with $k \leq \ell$, $h^{(k)}(t) = \mathrm{Coeff}_{Z^k}(h_\ell)$.

Proof. By the definition of Hasse derivative, $h^{(k)}(t) = \mathrm{Coeff}_{Z^k}(h(t + Z))$. On the other hand,

$$\begin{aligned} h(t + Z) &= f(g_1(t + Z), \ldots, g_n(t + Z)) \\ &= f(g_1 + Z\tilde{g}_1, \ldots, g_n + Z\tilde{g}_n). \end{aligned}$$

Applying Taylor's expansion on $f(g_1 + Z\tilde{g}_1, \ldots, g_n + Z\tilde{g}_n)$, we get that

$$\begin{aligned} h(t + Z) &= \sum_{i=0}^{d} Z^i \sum_{\mathbf{e} \in \mathbb{N}^n : |\mathbf{e}|_1 = i} \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{e}} \\ &= h_\ell + \sum_{i=\ell+1}^{d} Z^i \sum_{\mathbf{e} \in \mathbb{N}^n : |\mathbf{e}|_1 = i} \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{e}}. \end{aligned}$$

The lowest possible degree of Z in the second part of the above sum is greater than ℓ. Therefore, the coefficient of Z^k in the second part is zero since $k \leq \ell$. Hence,

$$h^{(k)}(t) = \mathrm{Coeff}_{Z^k}(h(t + Z)) = \mathrm{Coeff}_{Z^k}(h_\ell(t, Z)),$$

which completes the proof. □

6.1 A description of the algorithm

We start by describing the algorithm, followed by its analysis.

Algorithm 3 Efficient multivariate polynomial evaluation with large number of variables

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d and N points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$.
Output: $f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \ldots, f(\boldsymbol{\alpha}_N)$.

1: Let p be the characteristic of $\mathbb{F}_q$ and $q = p^a$.
2: Let $v_0(Y_0)$ be an irreducible polynomial in $\mathbb{F}_p[Y_0]$ of degree a and

$$\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle.$$

3: Let b the smallest positive integer such that $p^b > ad$.
4: Compute an irreducible polynomial $v_1(Y_1)$ in $\mathbb{F}_q[Y_1]$ of degree b and

$$\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle. \text{ (Lemma 4.2)}$$

5: Compute the subfield $\mathbb{F}_{p^b}$ of $\mathbb{F}_{q^b}$. (Lemma 4.4)
6: Compute the set $\overline{\partial}^{<n}(f)$. (Lemma 4.7)
7: Evaluate all the polynomials in $\overline{\partial}^{<n}(f)$ over the grid $\mathbb{F}_{p^b}^n$. (Lemma 4.11)
8: **for** all $i \in [N]$ **do**
9: Let $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1} Y_0 + \cdots + \boldsymbol{\alpha}_{i,a-1} Y_0^{a-1}$, where $\boldsymbol{\alpha}_{i,j} \in \mathbb{F}_p^n$.
10: Let $\mathbf{g}_i(t)$ be the curve defined as $\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1} t + \cdots + \boldsymbol{\alpha}_{i,a-1} t^{a-1}$.
11: Let $h_i(t) = f(\mathbf{g}_i(t))$.
12: Let $E_i = \{(\gamma, h_i^{(0)}(\gamma), h_i^{(1)}(\gamma), \ldots, h_i^{(n-1)}(\gamma) \mid \gamma \in \mathbb{F}_{p^b}\}$.
13: Invoke the function EVALUATE DERIVATIVES A with input $\mathbf{g}_i(t)$ and compute the set E_i.
14: Using E_i, interpolate $h_i(t)$. (Lemma 4.10)
15: Output $h_i(Y_0)$ as $f(\boldsymbol{\alpha}_i)$. (Lemma 4.8)

We now describe the function Evaluate Derivatives A invoked above. We follow the same notation as in Algorithm 5 including the local variable names.

Algorithm 4 Function to generate data for Hermite Interpolation

function EVALUATE DERIVATIVES A ($\mathbf{g}(t)$)
  Let $\mathbf{g}(t) = (g_1, \ldots, g_n)$.
  For all $i \in [n]$, let $g_i(t+Z) = g_i(t) + Z\tilde{g}_i(t,Z)$ and $\tilde{\mathbf{g}}(t,Z) = (\tilde{g}_1(t,Z), \ldots, \tilde{g}_n(t,Z))$.
  Compute $\tilde{g}_i(t,Z)$ for all $i \in [n]$. (Lemma 4.7)
  For all $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$, let $\tilde{\mathbf{g}}_{\mathbf{e}} = \prod_{i=1}^{n} \tilde{g}_i^{e_i}$.
  Compute the set of polynomials $\{\tilde{\mathbf{g}}_{\mathbf{e}}(t,Z) \mid |\mathbf{e}|_1 < n\}$. (Polynomial multiplication)
  $P \leftarrow \emptyset$.
  for all $\gamma \in \mathbb{F}_{p^b}$ **do**
    Using evaluations of polynomials in $\overline{\partial}^{<n}(f)$ over $\mathbb{F}_{p^b}^n$, compute the polynomial

$$h_\gamma(Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{e} \in \mathbb{N}^n : |e|_1 = i} \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}(\gamma))\tilde{\mathbf{g}}_{\mathbf{e}}(\gamma, Z).$$

    For all $i \in \{0, 1, \ldots, n-1\}$, extract $\mathrm{Coeff}_{Z^i}(h_\gamma)$.
    $P \leftarrow P \cup \{(\gamma, \mathrm{Coeff}_{Z^0}(h_\gamma), \mathrm{Coeff}_{Z^1}(h_\gamma), \ldots, \mathrm{Coeff}_{Z^{n-1}}(h_\gamma))\}$.
  return P.

6.2 Analysis of Algorithm 5

Proof of Theorem 6.1. We start with the proof of correctness.

Correctness of Algorithm 5. We show that Algorithm 5 computes $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in the desired time. Like Algorithm 2, we assume that the underlying field $\mathbb{F}_q$ is represented as $\mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$ for some degree a irreducible polynomial $v_0(Y_0)$ over $\mathbb{F}_p$. However, unlike Algorithm 5, here we pick b as the smallest positive integer satisfying $p^b > ad$. Like Algorithm 2, here also we construct a degree b extension $\mathbb{F}_{q^b}$ over $\mathbb{F}_q$ and compute all the elements of the subfield $\mathbb{F}_{p^b}$ (of $\mathbb{F}_{q^b}$). Lemma 4.2 ensures that we can construct $\mathbb{F}_{q^b}$ as $\mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$ for some degree b irreducible polynomial over $\mathbb{F}_q$ and from Lemma 4.4, we can compute $\mathbb{F}_{p^b}$. The crucial difference with Algorithm 2 is the way we interpolate the polynomial $h_i(t)$ in Line 14 of Algorithm 5. The field $\mathbb{F}_{p^b}$ may have much smaller number of points than the degree of $h_i(t)$. Therefore, to interpolate $h_i(t)$, we have to evaluate all the Hasse derivatives of $h_i(t)$ up to order $n-1$ at points in $\mathbb{F}_{p^b}$. Next we describe the correctness of Algorithm 5 in detail.

As mentioned in the proof of Theorem 5.1, the representation of $\mathbb{F}_q$ ensures that each $\boldsymbol{\alpha}_i$ is of form $\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}Y_0 + \cdots + \boldsymbol{\alpha}_{i,a-1}Y_0^{a-1}$ where $\boldsymbol{\alpha}_{i,j}$ is in $\mathbb{F}_p^n$. Therefore, $h_i(t) = f(\mathbf{g}_i(t))$ is a polynomial of degree less than adn since f is an n-variate polynomial with individual degree less than d. This implies that, like Algorithm 2, we can interpolate the $h_i(t)$ by evaluating it at adn many distinct

points. However, for the choice of b in this algorithm, we don't have adn elements. So, for all $\gamma \in \mathbb{F}_{p^b}$, we compute the evaluations at γ of all Hasse derivatives of h_i up to order $n-1$ and invoke Lemma 4.10 with this data to recover h_i. From Lemma 6.2, using the evaluations of $\overline{\partial}^{<n}(f)$ over the grid $\mathbb{F}_{p^b}^n$, the function EVALUATE DERIVATIVES A of Algorithm 4 computes the set E_i consisting of nth order derivative information of h_i for every $\gamma \in \mathbb{F}_{p^b}$. Given this set E_i, we invoke Lemma 4.10 to successfully interpolate $h_i(t)$ and output $f(\boldsymbol{\alpha}_i) = h_i(Y_0)$ for all $i \in [N]$.

Time complexity of Algorithm 5. We now describe the time complexity of Algorithm 5. Similar to Algorithm 2, using Lemma 4.2, the construction of $\mathbb{F}_{q^b}$ takes $\text{poly}(a, b, p)$ $\mathbb{F}_q$-operations and its basic operations can be done in $\text{poly}(b)$ $\mathbb{F}_q$-operations. From Lemma 4.4, all the elements of $\mathbb{F}_{p^b}$ can be computed in $p^b \cdot \text{poly}(a, b, p)$ $\mathbb{F}_q$-operations. Since $\binom{n+n-1}{n}$ is upper bounded by 4^n, applying Lemma 4.7, the set of polynomials $\overline{\partial}^{<n}(f)$ can be computed in $(4d)^n \cdot \text{poly}(d, n)$ $\mathbb{F}_q$-operations. Hence, from Lemma 4.11, computing all the polynomials in $\overline{\partial}^{<n}(f)$ over the grid $\mathbb{F}_{p^b}^n$ requires

$$(d^n + p^{bn}) \cdot 4^n \cdot \text{poly}(a, b, d, n, p)$$

$\mathbb{F}_q$-operations. Next we discuss the time taken by the *for* loop in Algorithm 5 at Line 8.

First we estimate the cost of each iteration of the loop. For that, we need to analyze the complexity of the function EVALUATE DERIVATIVES A. The input $\mathbf{g}(t) = (g_1, \ldots, g_n)$ to EVALUATE DERIVATIVES A is a curve of degree at most $a-1$. Using Lemma 4.7, $\tilde{\mathbf{g}}(t, Z) = (\tilde{g}_1, \ldots, \tilde{g}_n)$ can be computed in $\text{poly}(a, n)$ $\mathbb{F}_q$-operations. Thus, the total cost of computing the set $\{\tilde{\mathbf{g}}_{\mathbf{e}}(t, Z) \mid |\mathbf{e}|_1 < n\}$ is $4^n \cdot \text{poly}(a, n)$ $\mathbb{F}_q$-operations. Given $\gamma \in \mathbb{F}_{p^b}$, we can evaluate $\tilde{\mathbf{g}}_{\mathbf{e}}(t, Z)$ at $t = \gamma$ in $\text{poly}(a, b, n)$ $\mathbb{F}_q$-operations. Thus, for each $\gamma \in \mathbb{F}_{p^b}$, the polynomial $h_\gamma(Z)$ at Line 9 in Algorithm 4 can be computed at the cost of $4^n \cdot \text{poly}(a, b, n)$ $\mathbb{F}_q$-operations. After computing $h_\gamma(Z)$ as its list of coefficients, we collect the coefficients of Z^i of $h_\gamma(Z)$ for $i \in \{0, 1, \ldots, n-1\}$. This implies that each call of the function EVALUATE DERIVATIVES A performs $4^n \cdot p^b \cdot \text{poly}(a, b, n)$ $\mathbb{F}_q$-operations.

Now we return to analyzing the cost taken by each iteration of the *for* loop at Line 8 in Algorithm 5. From the above discussion, for each $i \in [N]$, the set E_i can be computed in $4^n \cdot p^b \cdot \text{poly}(a, b, n)$ $\mathbb{F}_q$-operations. Given E_i, applying Lemma 4.10, the interpolation of $h_i(t)$ requires $\text{poly}(a, b, d, n)$ operations in $\mathbb{F}_q$. Thus, each iteration of the *for* loop at Line 8 in Algorithm 5 takes $4^n \cdot p^b \cdot \text{poly}(a, b, d, n)$ $\mathbb{F}_q$-operations. Therefore, the total cost of the *for* loop is

$$N \cdot 4^n \cdot p^b \cdot \text{poly}(a, b, d, n)$$

$\mathbb{F}_q$-operations. Since $ad < p^b \leq adp$, combining the complexities of all the components, we get that Algorithm 5 performs

$$(N + (adp)^n) \cdot 4^n \cdot \text{poly}(a, d, n, p)$$

$\mathbb{F}_q$-operations. □

7 Multipoint evaluation with improved field dependence

In this section, we build on the ideas in Algorithm Theorem 6.1 to improve the dependence on the field size. Our main theorem, which is a formal statement of our main result Theorem 2.1 stated in the introduction.

Theorem 7.1. *Let p be a prime and $q = p^a$ for some positive integer a. There is a deterministic algorithm such that on input an n-variate polynomial $f(\mathbf{x})$ over $\mathbb{F}_q$ with individual degree less than d, points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$ and a non-negative integer $\ell \leq \log_p^\star(a)$, it outputs $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in time*

$$\left(N \cdot \left(2dp \log_p(dp)\right)^\ell + \left(2rdp \log_p(dp)\right)^n \right) \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, d, n, p)\,,$$

where $r = \max\{2, \log_p^{\circ \ell}(a)\}$.

7.1 A description of the algorithm

We start by describing the algorithm, followed by its analysis.

Algorithm 5 Efficient multivariate polynomial evaluation over large fields

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d, N points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$, and a non-negative integer $\ell \leq \log_p^\star(a)$ where $q = p^a$ and p is the characteristic of $\mathbb{F}_q$.

Output: $f(\boldsymbol{\alpha}_1), \ldots, f(\boldsymbol{\alpha}_N)$.

Let $v_0(Y_0)$ be an irreducible polynomial in $\mathbb{F}_p[Y_0]$ of degree a and $\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$.
$\mathsf{Points}_0 \leftarrow \{\boldsymbol{\alpha}_i \mid i \in [N]\}$, $a_0 \leftarrow a$, and $q_0 \leftarrow p^a$.
POLYNOMIAL EVALUATION(0). (Recursive call)
Output $\mathsf{Eval}_{0,\mathbf{0}}$.
function POLYNOMIAL EVALUATION(i)
    Let a_{i+1} be the smallest positive integer such that $p^{a_{i+1}} > a_i d$, and $q_{i+1} \leftarrow q^{a_{i+1}}$.
    Compute an irreducible polynomial $v_{i+1}(Y_{i+1})$ over $\mathbb{F}_q$ of degree a_{i+1} and

$$\mathbb{F}_{q_{i+1}} = \mathbb{F}_q[Y_{i+1}]/\langle v_{i+1}(Y_{i+1})\rangle. \text{ (Lemma 4.2)}$$

    Compute the subfield $\mathbb{F}_{p^{a_{i+1}}}$ of $\mathbb{F}_{q_{i+1}}$. (Lemma 4.4)
    Compute an element β_i in $\mathbb{F}_{q_i}$ s.t. $\{1, \beta_i, \ldots, \beta_i^{a_i-1}\}$ forms an $\mathbb{F}_p$-basis for $\mathbb{F}_{p^{a_i}}$. (Lemma 4.4)
    $\mathsf{Points}_{i+1} \leftarrow \emptyset$.
    for all $\boldsymbol{\alpha} \in \mathsf{Points}_i$ **do**
        Let $\boldsymbol{\alpha} = \boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1\beta_i + \cdots + \boldsymbol{\alpha}_{a_i-1}\beta_i^{a_i-1}$, where $\boldsymbol{\alpha}_j \in \mathbb{F}_p^n$.
        Compute $\boldsymbol{\alpha}_0, \ldots, \boldsymbol{\alpha}_{a_i-1}$. (Lemma 4.4)
        Let $\mathbf{g}_{\boldsymbol{\alpha}}(t)$ be the curve defined as $\boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1 t + \cdots + \boldsymbol{\alpha}_{a_i-1}t^{a_i-1}$.
        $P_{\boldsymbol{\alpha}} \leftarrow \{\mathbf{g}_{\boldsymbol{\alpha}}(\gamma) \mid \gamma \in \mathbb{F}_{p^{a_{i+1}}}\}$ (Lemma 4.8), and $\mathsf{Points}_{i+1} \leftarrow \mathsf{Points}_{i+1} \cup P_{\boldsymbol{\alpha}}$.
    if $i < \ell$ **then**
        POLYNOMIAL EVALUATION($i+1$).
    else
        Compute all the polynomials in $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$. (Lemma 4.7)
        Evaluate all the polynomials in $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$ over the grid $\mathbb{F}_{p^{a_{\ell+1}}}^n$. (Lemma 4.11)
        Observe that $\mathsf{Points}_{\ell+1}$ is a subset of $\mathbb{F}_{p^{a_{\ell+1}}}^n$.
        For all $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq (\ell+1)(n-1)$, $\mathsf{Eval}_{\ell+1,\mathbf{e}} = \{(\boldsymbol{\alpha}, \overline{\partial}_{\mathbf{e}}(f)(\boldsymbol{\alpha})) \mid \boldsymbol{\alpha} \in \mathbb{F}_{p^{a_{\ell+1}}}^n\}$.
    for all $\mathbf{e} \in \mathbb{N}^n$ s.t. $|\mathbf{e}|_1 \leq i(n-1)$ **do**
        $\mathsf{Eval}_{i,\mathbf{e}} \leftarrow \emptyset$.
        for all $\boldsymbol{\alpha} \in \mathsf{Points}_i$ **do**
            Let $h_{\mathbf{e},\boldsymbol{\alpha}}(t) = \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}_{\boldsymbol{\alpha}}(t))$.
            Let $E_{\mathbf{e},\boldsymbol{\alpha}} = \{(\gamma, h_{\mathbf{e},\boldsymbol{\alpha}}^{(0)}(\gamma), \ldots, h_{\mathbf{e},\boldsymbol{\alpha}}^{(n-1)}(\gamma)) \mid \gamma \in \mathbb{F}_{p^{a_{i+1}}}\}$.
            Using EVALUATE DERIVATIVES B with input $(\mathbf{g}_{\boldsymbol{\alpha}}(t), i, \mathbf{e})$, compute $E_{\mathbf{e},\boldsymbol{\alpha}}$.
            Using $E_{\mathbf{e},\boldsymbol{\alpha}}$, interpolate $h_{\mathbf{e},\boldsymbol{\alpha}}(t)$. (Lemma 4.10)
            $\mathsf{Eval}_{i,\mathbf{e}} \leftarrow \mathsf{Eval}_{i,\mathbf{e}} \cup \{(\boldsymbol{\alpha}, h_{\mathbf{e},\boldsymbol{\alpha}}(\beta_i))\}$. (Lemma 4.8)

Algorithm 6 Evaluating Hasse derivatives for Algorithm **??**

function EVALUATE DERIVATIVES B ($\mathbf{g}(t)$, k, $\mathbf{e}$)

Let $\mathbf{g}(t) = (g_1, \ldots, g_n)$.

Let $\mathbf{e} = (e_1, \ldots, e_n)$.

Let $g_i(t+Z) = g_i(t) + Z\tilde{g}_i(t,Z)$, for all $i \in [n]$, and $\tilde{\mathbf{g}}(t,Z) = (\tilde{g}_1(t,Z), \ldots, \tilde{g}_n(t,Z))$.

Compute $\tilde{g}_i(t,Z)$ for all $i \in [n]$. (Lemma 4.7)

For all $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$, let $\tilde{\mathbf{g}}_{\mathbf{b}} = \prod_{i=1}^{n} \tilde{g}_i^{b_i}$.

Compute the set of polynomials $\{\tilde{\mathbf{g}}_{\mathbf{b}}(t,Z) \mid |\mathbf{b}|_1 < n\}$. (Polynomial multiplication)

Let D be an $((k+1)(n-1)+1) \times n$ array such that,

$$D_{i,j} = \binom{j}{i} \bmod p, \text{ where } i \in \{0,1,\ldots,n-1\}, j \in \{0,1,\ldots,(k+1)(n-1)\}$$

Like Algorithm 1, we can compute D using $\mathbb{F}_p$-operations.

For $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$ such that $|\mathbf{b}|_1 < n$,

$$c_{\mathbf{b}} \leftarrow \prod_{i=1}^{n} D_{e_i+b_i, b_i}.$$

$P \leftarrow \emptyset$.

for all $\gamma \in \mathbb{F}_{p^{a_{k+1}}}$ **do**

Using evaluations of polynomials in $\overline{\partial}^{\leq (k+1)(n-1)}(f)$ over Points_{k+1}, compute

$$h_\gamma(Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{b} \in \mathbb{N}^n : |\mathbf{b}|_1 = i} c_{\mathbf{b}} \overline{\partial}_{\mathbf{e}+\mathbf{b}}(f)(\mathbf{g}(\gamma)) \tilde{\mathbf{g}}_{\mathbf{b}}(\gamma, Z).$$

For all $i \in \{0,1,\ldots,n-1\}$, extract $\mathrm{Coeff}_{Z^i}(h_\gamma)$.

$P \leftarrow P \cup \{(\gamma, \mathrm{Coeff}_{Z^0}(h_\gamma), \mathrm{Coeff}_{Z^1}(h_\gamma), \ldots, \mathrm{Coeff}_{Z^{n-1}}(h_\gamma))\}$.

return P.

7.2 Analysis of Algorithm ??

A useful lemma. The following lemma would be useful for the time complexity analysis of Algorithm **??**.

Lemma 7.2. *Let a, d and p be three positive integers such that p, d are greater than 1. Let $(a_0, a_1, a_2, \ldots)$ be a sequence of positive integers with the following properties: $a_0 = a$, and for all $i > 0$, a_i be the smallest positive integer such that $p^{a_i} > da_{i-1}$. Then for all non-negative integer i,*

$$a_i \leq 2r_i \log_p(dp),$$

where $r_i = \max\{2, \log_p^{\circ i}(a)\}$. *Furthermore, for any non-negative integer* $\ell \leq \log_p^{\star}(a)$,

$$\prod_{i=0}^{\ell} a_i \leq (2\log_p(dp))^{\ell} \cdot a^{1+o(1)}.$$

Proof. From the definition of the sequence, it is not hard to see that for every positive integer i, $p^{a_i} \leq dpa_{i-1}$. We inductively show that for every non-negative integer i, $a_i \leq 2r_i \log_p(dp)$. It is true for $i = 0$ since $\log_p^{\circ i}(a) = a$ for $i = 0$ and $\log_p(dp) \geq 1$. This establishes our base case. Now assume that $a_i \leq 2r_i \log_p(dp)$ for some integer $i \geq 0$. We know that $p^{a_{i+1}} \leq dpa_i$. Taking logarithm with respect to p, we get that $a_{i+1} \leq \log_p(dp) + \log_p(a_i)$. From the induction hypothesis, we get that

$$\begin{aligned} a_{i+1} &\leq \log_p(dp) + \log_p(2r_i \log_p(dp)) \\ &= \log_p(dp) + \log_p \log_p(dp)^2 + \log_p(r_i) \end{aligned}$$

From the definition of r_{i+1}, we get that $r_{i+1} \geq \log_p(r_i)$. Also, $(dp)^2 \leq p^{dp}$ for all $d, p \geq 2$. Therefore, $a_{i+1} \leq 2\log_p(dp) + r_{i+1}$. Since both $2\log_p(dp)$ and r_{i+1} are ≥ 2,

$$2\log_p(dp) + r_{i+1} \leq 2r_{i+1}\log_p(dp).$$

This completes the induction step.

Now we prove the second part of the above lemma. From the first half, we get that

$$\prod_{i=0}^{\ell} a_i \leq a(2\log_p(dp))^{\ell} \cdot \prod_{i=1}^{\ell} r_i.$$

Let k be the largest non-negative integer such that $\log_p^{\circ k}(a) \geq 2$. First, assume that $\ell \leq k$. Then

$$\begin{aligned} \prod_{i=0}^{\ell} a_i &\leq a(2\log_p(dp))^{\ell} \cdot \prod_{i=1}^{\ell} r_i \\ &\leq a(2\log_p(dp))^{\ell} \cdot \prod_{i=1}^{\ell} \log_p^{\circ i}(a) \\ &\leq (2\log_p(dp))^{\ell} \cdot a^{1+o(1)}. \end{aligned}$$

Since $\log_p^{\star}(a)$ can be at most $k+2$, $\ell \leq k+2$. Now, assume that $\ell > k$.

$$\prod_{i=0}^{\ell} a_i \leq a(2\log_p(dp))^{\ell} \cdot \prod_{i=1}^{\ell} r_i$$

$$\leq a(2\log_p(dp))^{\ell} \cdot \left(\prod_{i=1}^{k} \log_p^{\circ i}(a)\right) \cdot 2^{\ell-k}$$
$$\leq (2\log_p(dp))^{\ell} \cdot a^{1+o(1)}.$$

□

We are now ready to discuss the proof of Theorem 7.1. As discussed in section 3, the main idea in reducing the dependence of the running time on the underlying field is to reduce the question of multipoint evaluation over the field $\mathbb{F}_{p^{a_0}}$ to an instance of multipoint evaluation with a larger number of points, but all these points lie in a smaller field $\mathbb{F}_{p^{a_1}}$, where $p^{a_1} \geq a_0 dn$. This reduction in the size leads to a significant decrease in the degree of the curves used in the local computation step, at the cost of increasing the number of points. We now make this intuition formal, and prove the necessary quantitative bounds.

Proof of Theorem 7.1. We start with a proof of correctness.

Correctness of Algorithm ??. We prove that Algorithm **??** computes $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ with the desired time complexity. First, we briefly highlight the main difference between Algorithm **??** and Algorithm 5. In Algorithm 5, we construct the field $\mathbb{F}_{q^b}$, a degree b extension of $\mathbb{F}_q$, such that both $\mathbb{F}_q$ and $\mathbb{F}_{p^b}$ are its subfields and $p^b > ad$. Next, we evaluate all the polynomials in $\overline{\partial}^{\leq n-1}(f)$ over the grid $\mathbb{F}_{p^b}^n$. Finally, we reduce the evaluation of $f(\mathbf{x})$ at points in $\mathbb{F}_q^n$ to the evaluation of the polynomials in $\overline{\partial}^{\leq n-1}(f)$ at the points in $\mathbb{F}_{p^b}^n$, and use the evaluations of $\overline{\partial}^{\leq n-1}(f)$ at the grid $\mathbb{F}_{p^b}^n$ to compute $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$. In short, Algorithm 5 reduces the evaluation of $f(\mathbf{x})$ at points in $\mathbb{F}_q^n$ to the evaluation of $\overline{\partial}^{\leq n-1}(f)$ at the points in a "smaller" grid $\mathbb{F}_{p^b}^n$. In Algorithm **??**, we repeat this reduction $(\ell+1)$ times where at the ith iteration, we reduce the question of evaluating the set of all Hasse derivatives of f of order up to $i(n-1)$ on a subset of points(Points_i) in $\mathbb{F}_{p^{a_i}}^n$ to the question of evaluating all the Hasse derivatives of f of order up to $(i+1)(n-1)$ on a subset of points(Points_{i+1}) in $\mathbb{F}_{p^{a_{i+1}}}^n$. Finally, at the ℓth iteration, we reach a much "smaller" grid $\mathbb{F}_{p^{a_{\ell+1}}}^n$ (compare to $\mathbb{F}_q^n$) where we evaluate all the Hasse derivatives of $f(\mathbf{x})$ up to order $(\ell+1)(n-1)$. We now show via an induction on i, with i decreasing from ℓ to 0, the following claim holds.

Claim 7.3. *For every $i \in \{0, 1, \ldots, \ell\}$, at the end of the function call* POLYNOMIAL EVALUATION (i), *we have correctly computed the evaluation of all polynomials in the set $\overline{\partial}^{\leq i(n-1)}(f)$ at all points in the set* Points_i.

Recall that the set $\mathsf{Points}_0 = \{\boldsymbol{\alpha}_i \mid i \in [N]\}$ is the original set of input points and hence, this suffices for the correctness of the algorithm. To proceed with the induction, we need the following subclaim whose proof we defer to the end. Recall the set $E_{\mathbf{e},\boldsymbol{\alpha}}$ defined in Line 27 of Algorithm **??**.

Claim 7.4. *Given* $(\mathbf{g}_{\boldsymbol{\alpha}}(t), i, \mathbf{e})$ *as the input, the function* EVALUATE DERIVATIVES B *of Algorithm 6 computes the set* $E_{\mathbf{e},\boldsymbol{\alpha}}$ *in time*

$$4^n \cdot \mathrm{poly}(a_i, a_{i+1}, d, n, p)\,.$$

We now proceed with the inductive proof of Claim 7.3.

Base case. For $i = \ell$, in Line 22 of Algorithm **??**, for every $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq (\ell+1)(n-1)$, we first compute the set $\mathsf{Eval}_{\ell+1,\mathbf{e}}$ which is the evaluation table of the polynomial $\overline{\partial}_{\mathbf{e}}(f)$ at all points in the set $\mathbb{F}^n_{p^{a_{\ell+1}}}$ via a multidimensional FFT (Lemma 4.11). We now claim that at the end of the subsequent *for* loop (Lines $23-30$), for every $\boldsymbol{\alpha} \in \mathsf{Points}_\ell$ and for every $\mathbf{e}$ in $\mathbb{N}^n$ with $|\mathbf{e}|_1 \leq \ell(n-1)$, we have the value of $\overline{\partial}_{\mathbf{e}}(f)$ at $\boldsymbol{\alpha}$. For this, consider a point $\boldsymbol{\alpha} \in \mathsf{Points}_\ell$ and $\mathbf{e} \in \mathbb{N}^n$ such that $|\mathbf{e}|_1 \leq \ell(n-1)$. Now the curve $\mathbf{g}_{\boldsymbol{\alpha}}(t)$ can be computed efficiently as in the proofs of correctness of Algorithm 2 and Algorithm 5. Then, $h_{\mathbf{e},\boldsymbol{\alpha}} = \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}_{\boldsymbol{\alpha}}(t))$ is a polynomial of degree less than $a_\ell dn$. From Claim 7.4, we have that the function EVALUATE DERIVATIVES B correctly computes the set $E_{\mathbf{e},\boldsymbol{\alpha}}$. Moreover, by our choice of $a_i's$, we have that $p^{a_{\ell+1}} \geq a_\ell d$. Thus, from Lemma 4.10, we can interpolate $h_{\mathbf{e},\boldsymbol{\alpha}}(t)$ correctly from the set $E_{\mathbf{e},\boldsymbol{\alpha}}$.

Induction step. In the induction step, we assume Claim 7.3 is true for $i = i_0 \leq \ell$ and show that it holds for the iteration $i_0 - 1$. The proof of this is precisely the same as that of the base case. The only difference is that in the base case, the set $\mathsf{Eval}_{\ell+1,\mathbf{e}}$ for every $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq (\ell+1)(n-1)$ was computed in Line 22 of the algorithm directly via the multidimensional FFT. In the induction step, for iteration $i_0 - 1$, we need the corresponding set $\mathsf{Eval}_{i_0,\mathbf{e}}$ for every $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq i_0(n-1)$. Note that these are sets of evaluations of all derivatives of order at most $i_0(n-1)$ of f on the point set Points_{i_0} that are guaranteed to be available to us by the induction hypothesis. The rest of the argument is exactly as that in the base case. We skip the details.

A point to note for the proof of both the base case and the induction step of Claim 7.3, is that $h_{\mathbf{e},\boldsymbol{\alpha}}$ is a polynomial with coefficients in $\mathbb{F}_q$ and $\mathbb{F}_q$ is a subfield of $\mathbb{F}_{q_i}$ for every $i \in \{0, 1, \ldots, \ell\}$, so all the algebra is consistent here.

This completes the proof of Claim 7.3 and hence the correctness of **??** (which follows from $i = 0$ case of the claim), modulo the proof of Claim 7.4. We defer that to the end of this section, and discuss the time complexity of Algorithm **??**.

Time complexity of Algorithm ??. Now we discuss the time complexity of Algorithm **??**. We rely on the following claim, whose proof we defer to the end of this section.

Claim 7.5. $|\mathsf{Points}_\ell| \leq N \cdot (2dp \log_p(dp))^\ell \cdot a^{1+o(1)}$.

For all $i \in \{0, 1, \ldots, \ell\}$, let T_i be the time complexity of the ith invocation of the function POLYNOMIAL EVALUATION. Next, we discuss the various components of T_i.

1. Using Lemma 4.2, the field $\mathbb{F}_{q_{i+1}} = \mathbb{F}_q[Y_{i+1}]/\langle v_{i+1}(Y_{i+1})\rangle$ can be constructed in $\mathrm{poly}(a, a_{i+1}, p)$ $\mathbb{F}_q$-operations. From Lemma 4.4, the cost of computing all the elements of the subfield $\mathbb{F}_{p^{a_{i+1}}}$ (of $\mathbb{F}_{q_{i+1}}$) is $p^{a_{i+1}} \cdot \mathrm{poly}(a, a_{i+1}, p)$ $\mathbb{F}_q$-operations.

2. Using Lemma 4.4, we can also compute the element β_i in $\mathrm{poly}(a, a_i, p)$ $\mathbb{F}_q$-operations. In addition, Lemma 4.4 ensures that for every $\gamma \in \mathbb{F}_{p^{a_i}}$, the $\mathbb{F}_p$-linear combination of γ with respect to $\{1, \beta_i, \ldots, \beta_i^{a_i-1}\}$ can be computed using $\mathrm{poly}(a_i)$ $\mathbb{F}_q$-operations. Therefore, for every $\boldsymbol{\alpha} \in \mathsf{Points}_i$, the cost of computing the curve $\mathbf{g}_{\boldsymbol{\alpha}}(t)$ is $\mathrm{poly}(a_i, n)$ $\mathbb{F}_q$-operations. This implies that the set $P_{\boldsymbol{\alpha}}$ can be constructed in $p^{a_{i+1}} \cdot \mathrm{poly}(a_i, a_{i+1}, n)$ $\mathbb{F}_q$-operations. Thus, the total cost of computing the set Points_{i+1} is

$$|\mathsf{Points}_i| \cdot p^{a_{i+1}} \cdot \mathrm{poly}(a_i, a_{i+1}, n)$$

   $\mathbb{F}_q$-operations.

3. For $i \in \{0, 1, \ldots, \ell-1\}$, we need to add the cost of $(i+1)$th call of Polynomial Evaluation, that is T_{i+1}. For $i = \ell$, using Lemma 4.7, we can compute the set $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$ in $\binom{(\ell+1)(n-1)+n}{n} \cdot d^n \cdot \mathrm{poly}(n)$ many $\mathbb{F}_q$-operations. Since $\binom{(\ell+1)(n-1)+n}{n} \leq O(\ell+1)^n$, the total cost of computing the set $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$ is

$$O(\ell+1)^n \cdot d^n \cdot \mathrm{poly}(n)$$

   $\mathbb{F}_q$-operations. We have to evaluate all the polynomials in $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$ over the grid $\mathbb{F}^n_{p^{a_{\ell+1}}}$. Using Lemma 4.11, each polynomial in $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$ can be evaluated over the grid in

$$(d^n + p^{na_{\ell+1}}) \cdot \mathrm{poly}(a, a_{\ell+1}, d, n, p)$$

   many $\mathbb{F}_q$-operations. Thus, the total cost of evaluating all the polynomials in $\overline{\partial}^{\leq(\ell+1)(n-1)}(f)$ over the grid is

$$(a_\ell dp)^n \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, a_{\ell+1}, d, n, p)$$

   $\mathbb{F}_q$-operations since $p^{a_{\ell+1}} \leq a_\ell dp$.

4. We have to compute the set $\mathsf{Eval}_{i,\mathbf{e}}$ for all $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq i(n-1)$. Let $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq i(n-1)$, and $\boldsymbol{\alpha} \in \mathsf{Points}_i$. Then, using Claim 7.4, the set $E_{\mathbf{e},\boldsymbol{\alpha}}$ can be computed in $4^n \cdot \mathrm{poly}(a_i, a_{i+1}, d, n, p)$ many $\mathbb{F}_q$-operations. Applying Lemma 4.10, the polynomial $h_{\mathbf{e},\boldsymbol{\alpha}}(t)$ can be interpolated from $E_{\mathbf{e},\boldsymbol{\alpha}}$ using $\mathrm{poly}(a_i, a_{i+1}, d, n)$ $\mathbb{F}_q$-operations. Finally, $h_{\mathbf{e},\boldsymbol{\alpha}}(\beta_i)$ can be computed in $\mathrm{poly}(a_i, d, n)$ $\mathbb{F}_q$-operations. Thus, for an $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq i(n-1)$, the cost of computing $\mathsf{Eval}_{i,\mathbf{e}}$ is $|\mathsf{Points}_i| \cdot 4^n \cdot (a_i, a_{i+1}, d, n, p)$ $\mathbb{F}_q$-operations. The number of $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq i(n-1)$ is $\binom{i(n-1)+n}{n}$, and it is upper bounded by $O(i+1)^n$. Therefore, the total

of computing $\mathsf{Eval}_{i,\mathbf{e}}$ for all $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \leq i(n-1)$ is

$$|\mathsf{Points}_i| \cdot O(i+1)^n \cdot \mathrm{poly}(a_i, a_{i+1}, d, n, p)$$

$\mathbb{F}_q$-operations.

The choice of a_{i+1} gives us that $p^{a_{i+1}} \leq a_i dp$. From Lemma 7.2, $a_i \leq 4a \log_p(dp)$ for all $i \geq 1$. Thus, from the above discussion, for all $i \in \{0, 1, \ldots, \ell-1\}$,

$$T_i \leq T_{i+1} + |\mathsf{Points}_i| \cdot O(i+1)^n \cdot \mathrm{poly}(a, d, n, p).$$

Also, the complexity of the ℓth invocation of POLYNOMIAL EVALUATION is

$$T_\ell \leq (a_\ell dp)^n \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, d, n, p).$$

Therefore, the overall $\mathbb{F}_q$-operations performed by Algorithm **??** is

$$\begin{aligned} T_0 &\leq \sum_{i=0}^{\ell-1} |\mathsf{Points}_i| \cdot O(i+1)^n \cdot \mathrm{poly}(a, d, n, p) + T_\ell \\ &\leq \ell |\mathsf{Points}_\ell| \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, d, n, p) + (a_\ell dp)^n \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, d, n, p) \\ &\leq (|\mathsf{Points}_\ell| + (a_\ell dp)^n) \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, d, n, p). \end{aligned}$$

Using Lemma 7.2, $a_\ell \leq 2r \log_p(dp)$, where $r = \max\{2, \log_p^{\circ \ell}(a)\}$. From Claim 7.5,

$$|\mathsf{Points}_\ell| \leq N \cdot (2dp \log_p(dp))^\ell \cdot a^{1+o(1)}.$$

Therefore, the number of $\mathbb{F}_q$-operations performed by Algorithm **??**

$$T_0 \leq \left(N \cdot \left(2dp \log_p(dp)\right)^\ell + \left(2rdp \log_p(dp)\right)^n \right) \cdot O(\ell+1)^n \cdot \mathrm{poly}(a, d, n, p).$$

□

This completes the proof of Theorem 7.1 modulo the proofs of the Claim 7.4 and Claim 7.5. We now discuss these missing proofs.

Proof of Claim 7.4

Proof of Claim 7.4. According to the function call, $\mathbf{g}(t) = \mathbf{g}_{\boldsymbol{\alpha}}(t)$ and $k = i$. Also, $\mathbf{g}(t) = (g_1, \ldots, g_n)$, $g_i(t+Z) = g_i(t) + Z\tilde{g}_i(t, Z)$ for all $i \in [n]$ and for all $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$, $\tilde{\mathbf{g}}_{\mathbf{b}} = \prod_{i=1}^n \tilde{g}_i^{b_i}$. Let

$\overline{\partial}_{\mathbf{e}}(f) = f_{\mathbf{e}}$. Let

$$h(t+Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{b}\in\mathbb{N}^n:|\mathbf{b}|_1\leq n-1} \overline{\partial}_{\mathbf{b}}(f_{\mathbf{e}})(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{b}}.$$

Then, from Proposition 4.6,

$$h(t+Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{b}\in\mathbb{N}^n:|\mathbf{b}|_1\leq n-1} \binom{\mathbf{e}+\mathbf{b}}{\mathbf{b}} \overline{\partial}_{\mathbf{e}+\mathbf{b}}(f)(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{b}}.$$

In step 10 of Algorithm 6, for all $\mathbf{b} \in \mathbb{N}^n$ with $|\mathbf{b}|_1 \leq n-1$, $c_{\mathbf{b}} = \binom{\mathbf{e}+\mathbf{b}}{\mathbf{b}}$. Therefore,

$$h(t+Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{b}\in\mathbb{N}^n:|\mathbf{b}|_1\leq n-1} c_{\mathbf{b}} \overline{\partial}_{\mathbf{e}+\mathbf{b}}(f)(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{b}}.$$

Applying Lemma 6.2, we get that for all $i \in \{0, 1, \ldots, n-1\}$, the ith order Hasse derivative of $f_{\mathbf{e}}(\mathbf{g}(t))$ is same as $\text{Coeff}_{Z^i}(h(t+Z))$. Hence, for all $\gamma \in \mathbb{F}_{p^{a_{k+1}}}$, the evaluation of ith order Hasse derivative of $f_{\mathbf{e}}(\mathbf{g}(t))$ at γ is equal to $\text{Coeff}_{Z^i}(h(\gamma+Z))$. In terms of notation used in Algorithm 6, $\text{Coeff}_{Z^i}(h(\gamma+Z))$ is same as $\text{Coeff}_{Z^i}(h_\gamma(Z))$. This implies that the evaluation of ith order Hasse derivative of $f_{\mathbf{e}}(\mathbf{g}(t))$ at γ is equal to $\text{Coeff}_{Z^i}(h_\gamma(Z))$. This implies that given $(\mathbf{g}_{\boldsymbol{\alpha}}(t), k, \mathbf{e})$ as input, the set P returned by EVALUATE DERIVATIVES B is same as $E_{\mathbf{e},\boldsymbol{\alpha}}$. Now, to compute $h_\gamma(Z)$, we need access to $\tilde{\mathbf{g}}_{\mathbf{b}}(\gamma, Z)$ for all $\mathbf{b} \in \mathbb{N}^n$ with $|\mathbf{b}|_1 < n$. Lemma 4.7 ensures that we can compute $\tilde{g}_i(t, Z)$ for all $i \in [N]$. After we have all $\tilde{g}_i(t, Z)$'s, we can compute $\tilde{\mathbf{g}}_{\mathbf{b}}(t, Z)$ and evaluate it at $t = \gamma$. Also, we need the access of $\overline{\partial}_{\mathbf{e}+\mathbf{b}}(f)(\mathbf{g}(\gamma))$ for all $\mathbf{b} \in \mathbb{N}^n$ with $|\mathbf{b}|_1 < n$. Observe that $|\mathbf{e}+\mathbf{b}|_1 \leq (k+1)(n-1)$ and $\mathbf{g}(\gamma) \in \text{Points}_{k+1}$. Therefore, from the evaluations of the polynomials $\overline{\partial}^{\leq(k+1)(n-1)}(f)$ at points Points_{k+1}, we get $\overline{\partial}_{\mathbf{e}+\mathbf{b}}(f)(\mathbf{g}(\gamma))$.

Now we discuss the number of $\mathbb{F}_q$-operations performed by Algorithm 6. From Lemma 4.7, we can compute $\tilde{g}_i(t, Z)$ for all $i \in [n]$ in $\text{poly}(a_k, n)$ $\mathbb{F}_q$-operations. Since each $\tilde{g}_i(t, Z)$ is a bivariate polynomial of individual degree less than a_k and $\binom{n+n-1}{n} \leq 4^n$, we can compute the set $\{\tilde{\mathbf{g}}_{\mathbf{b}}(t, Z) \mid |\mathbf{b}|_1 < n\}$ in $4^n \cdot \text{poly}(a_k, n)$ $\mathbb{F}_q$-operations. Computing all $c_{\mathbf{b}}$'s takes $4^n \cdot \text{poly}(n) + \text{poly}(k, n)$ $\mathbb{F}_q$-operations. Given a $\gamma \in \mathbb{F}_{p^{a_{k+1}}}$, from Lemma 4.4, we can evaluate $\tilde{\mathbf{g}}_{\mathbf{b}}$ at $t = \gamma$ in $\text{poly}(a_k, a_{k+1}, n)$ $\mathbb{F}_q$-operations. Thus, for any $\gamma \in \mathbb{F}_{p^{a_{k+1}}}$, the cost of computing $h_\gamma(Z)$ is $4^n \cdot \text{poly}(a_k, a_{k+1}, n)$. Once we have $h_\gamma(Z)$ as its list of coefficients, we collect the coefficients of Z^i for all $i \in \{0, 1, \ldots, n-1\}$. From the choice of a_{k+1}, $p^{a_{k+1}} \leq a_k dp$. Thus, the total cost of Algorithm 6 is

$$4^n \cdot \text{poly}(a_k, a_{k+1}, d, p, n)$$

$\mathbb{F}_q$-operations. □

Proof of Claim 7.5

Proof of Claim 7.5. First, we show that for all $i \in \{0, 1, \ldots, \ell - 1\}$, $|\mathsf{Points}_{i+1}| \leq |\mathsf{Points}_i| \cdot (a_i dp)$. From the step 15 of Algorithm **??**, the set

$$\mathsf{Points}_{i+1} = \cup_{\boldsymbol{\alpha} \in \mathsf{Points}_i} P_{\boldsymbol{\alpha}}.$$

The definition of $P_{\boldsymbol{\alpha}}$ ensures that its size is at most $p^{a_{i+1}}$, which is upper bounded by $a_i dp$. Therefore, the size of Points_{i+1} is at most $|\mathsf{Points}_i| \cdot (a_i dp)$. This implies that

$$\mathsf{Points}_\ell \leq N \cdot (dp)^\ell \cdot \prod_{i=0}^{\ell} a_i.$$

From Lemma 7.2, $\prod_{i=0}^{\ell} a_i \leq (2 \log_p(dp))^\ell \cdot a^{1+o(1)}$. Therefore,

$$|\mathsf{Points}_\ell| \leq N \cdot (2dp \log_p(dp))^\ell \cdot a^{1+o(1)}.$$

□

8 An algebraic data structure for polynomial evaluation

In this section, we discuss the implication of our multipoint evaluation algorithms to the question of data structures for polynomial evaluation. For functions $s : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N} \to \mathbb{N}$, an algebraic data structure for univariate polynomial evaluation over a field $\mathbb{F}$ with space complexity $s(n)$ and time complexity $t(n)$ is specified by a preprocessing map and a query algorithm. For every $n \in \mathbb{N}$, the preprocessing map maps a polynomial $f \in \mathbb{F}[X]$ of degree less than n to an $s(n)$ dimensional vector over $\mathbb{F}$ which we denote by $\mathcal{D}_f$ and the query algorithm is an algebraic algorithm that on any input $\alpha \in \mathbb{F}$, accesses at most $t(n)$ coordinates of $\mathcal{D}_f$ and correctly outputs $f(\alpha)$.

For this discussion $\mathbb{F}$ is a finite field, and as mentioned in Remark 2.2, we assume that the query algorithm has access to a description of the field, for instance via an irreducible polynomial of appropriate degree over the base field. Now, we formally state the main result for this section.

Theorem 8.1. *Let p be a fixed prime. Then, for all sufficiently large $n \in \mathbb{N}$ and all fields $\mathbb{F}_{p^a}$ with $a = \mathrm{poly}(\log n)$, there is an algebraic data structure for polynomial evaluation for univariate polynomials of degree less than n over $\mathbb{F}_{p^a}$ that has space complexity at most $n^{1+o(1)}$ and query complexity at most $n^{o(1)}$.*

Moreover, given a description of $\mathbb{F}_{p^a}$ (via an irreducible polynomial of degree a over $\mathbb{F}_p$) there is an algebraic algorithm that when given the coefficients of a univariate polynomial f of degree less than n computes the output of the preprocessing map on f, denoted here by $\mathcal{D}_f$ in time $n^{1+o(1)}$ and an algebraic algorithm which, when given an $\alpha \in \mathbb{F}_{p^a}$ and $\mathcal{D}_f$, outputs $f(\alpha)$ in time $n^{o(1)}$.

We recall that for fields of small characteristic and size $\text{poly}(n)$, Theorem 8.1 provides a counterexample to a conjecture of Milterson from [Mil95] that any algebraic data structure for polynomial evaluation over small fields that has space complexity $\text{poly}(n)$ must have query complexity linear in n. We also note that a slightly more general version of Theorem 8.1 is true where we have an appropriate tradeoff between the query and the space complexities. However, for the ease of exposition, we focus on proving the specific statement in Theorem 8.1.

8.1 Proof of Theorem 8.1

The proof is a very simple application of the ideas in the multipoint evaluation algorithms discussed in the earlier section. The first ingredient is a reduction from the univariate problem to the multivariate problem. This step is also there in the data structure of Kedlaya & Umans [KU11].

Definition 8.2 (Inverse Kronecker Map)**.** *Let $\mathbb{F}$ be a field. Then, for parameters $d, m \in \mathbb{N}$, the map $\psi_{d,m}$ from $\mathbb{F}[X]$ to $\mathbb{F}[Z_1, \ldots, Z_m]$ is defined as follows: Given a monomial X^t, write t in base d, $t = \sum_{j \geq 0} t_j d^j$ and define the monomial*

$$M_a(\mathbf{z}) := Z_1^{t_0} Z_2^{t_1} \cdots Z_m^{t_{m-1}}.$$

The map $\psi_{d,m}$ sends X^a to $M_a(\mathbf{z})$ and extends multilinearly to $\mathbb{F}[X]$. ⌟

The map $\psi_{d,m}(f)$ can be computed in linear time in the size of f, assuming f is represented explicitly by its coefficients. Also, $\psi_{d,m}$ is injective on the polynomials of degree less than d^m. For such polynomial f, if $F = \psi_{d,m}(f)$, then

$$f(X) = F(X^{d^0}, X^{d^1}, \ldots, X^{d^{m-1}}).$$

Given a degree $n-1$ univariate polynomial f, let the parameters m, d be set as follows: $m = \log\log n$ and $d = n^{1/\log\log n}$ and construct the polynomial $F = \psi_{d,m}(f)$ on m variables and degree at most $d-1$ in each variable. Clearly, this can be done in time $n \cdot \text{poly}(\log n)$ by processing f one monomial at a time. We now describe the preprocessing map and its image on f denoted $\mathcal{D}_f$ using this polynomial F. This construction is based on the Theorem 5.1. A slightly more general statement can be obtained by relying on the more involved algorithms for multipoint evaluation in section 6 and section 7, but for the proof of Theorem 8.1, Algorithm 2 is sufficient.

The preprocessing map. Let $v_0(Y_0)$ be an irreducible polynomial over $\mathbb{F}_p$ of degree a such that $\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$, where $q = p^a$. We assume that this v_0 is given as a part of the input, since it conveys the description of the field we are working over. Compute the smallest integer of form p^b such that $p^b > adm$. Compute an irreducible polynomial $v_1(Y_1)$ in $\mathbb{F}_q[Y_1]$ of degree b such that $\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$. Using Lemma 4.4, compute the subfield $\mathbb{F}_{p^b}$ of $\mathbb{F}_{q^b}$. We compute and store the evaluation of F on every input in the grid $\mathbb{F}_{p^b}^m$. This is our $\mathcal{D}_f$. The space required to store all

this data is at most $(m+1)p^{bm}$ elements of the field $\mathbb{F}_{p^{ab}}$ of equivalently $(m+1)p^{bm}b$ elements of the field $\mathbb{F}_{p^a}$. For our choice of parameters, the space complexity can be upper bounded as follows.

$$p^{bm} \leq (padm)^m = p^{\log\log n} \cdot (\text{poly}(\log n))^{\log\log n} \cdot n \cdot (\log\log n)^{\log\log n} \leq n^{1+o(1)}.$$

Thus, the space complexity is at most $(m+1)p^{bm}b \leq n^{1+o(1)}$ as claimed. Moreover, $\mathcal{D}_f$ can be computed by an algebraic algorithm over $\mathbb{F}_q$ using Lemma 4.11 and other ideas in section 6.

Answering evaluation queries using $\mathcal{D}_f$. We now describe an algorithm that given an $\alpha \in \mathbb{F}_q$ and access to $\mathcal{D}_f$ computes $f(\alpha)$ in time $n^{o(1)}$. The algorithm is essentially the same as the local computation step in Algorithm 2. Note that $\mathcal{D}_f$ contains precisely the data that is computed in Algorithm 2 in the preprocessing phase. We assume the notation (F, v_0, v_1 etc.) set up in the previous paragraph.

Algorithm 7 Evaluating polynomial $f(x)$ at a point in $\mathbb{F}_q$ using $\mathcal{D}_f$

Input: A point $\alpha \in \mathbb{F}_q$ and query access to $\mathcal{D}_f$.
Output: $f(\alpha)$.

1: Let $\boldsymbol{\alpha} = (\alpha^{d^0}, \alpha^{d^1}, \ldots, \alpha^{d^{n-1}})$.
2: Let $\boldsymbol{\alpha} = \boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1 Y_0 + \cdots + \boldsymbol{\alpha}_{a-1} Y_0^{a-1}$, where $\boldsymbol{\alpha}_j \in \mathbb{F}_p^n$.
3: Compute $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{k-1}$.
4: Let $\mathbf{g}(t)$ be the curve defined as $\boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1 t + \cdots + \boldsymbol{\alpha}_{a-1} t^{a-1}$.
5: Compute the set $P = \{(\gamma, \mathbf{g}(\gamma)) \mid \gamma \in \mathbb{F}_{p^b}\}$.
6: Collect the set $E = \{(\gamma, F(\boldsymbol{\gamma}')) \mid (\gamma, \boldsymbol{\gamma}') \in P\}$ by querying $\mathcal{D}_f$.
7: Using E, interpolate the univariate polynomial $F(\mathbf{g}(t))$.
8: Output $F(\mathbf{g}(Y_0))$ as $f(\alpha)$.

The proof of correctness of the construction of $\mathcal{D}_f$ and the query algorithm immediately follow from the proof of correctness of Algorithm 2. The query complexity is clearly upper bounded by $p^b \leq padm$, which for our setting of parameters, i.e. $p = O(1)$, $a = \text{poly}(\log n)$, $d = n^{1/\log\log n}$ and $m = \log\log n$ is $n^{o(1)}$. This completes the proof of Theorem 8.1.

9 Rigidity uppper bounds

In this section, we prove Theorem 2.6. We start with the definition of matrix rigidity which was introduced by Valiant [Val77].

Definition 9.1. *(Matrix rigidity) For a matrix M over some field $\mathbb{F}$ and a natural number r, we define $R_M^{\mathbb{F}}(r)$ to be the smallest number s for which there exists a matrix A with at most s nonzero entries and*

a matrix B of rank at most r such that $M = A + B$. If $R_M^{\mathbb{F}}(r) \geq s$, we say M is (r, s)-rigid. When the underlying field is clear from the context, we drop the superscript and denote $R_M^{\mathbb{F}}(r)$ by $R_M(r)$. ⌟

Now we define *regular rigidity* denoted by $r_M^{\mathbb{F}}(\cdot)$.

Definition 9.2. *(Regular rigidity) For a matrix M over some field $\mathbb{F}$ and a natural number $\tilde{r}$, we define $r_M(\tilde{r})$ to be the smallest number s such that there exists a matrix A with at most s nonzero entries in each row and column and a matrix B of rank at most $\tilde{r}$ such that $M = A + B$. If $r_M^{\mathbb{F}}(\tilde{r}) \geq s$, we say M is $(\tilde{r}, s)$-regular rigid. Also here, when the underlying field is clear from the context, we drop the superscript and denote $r_M^{\mathbb{F}}(\tilde{r})$ by $r_M(\tilde{r})$.* ⌟

Note that the notion of *regular rigidity* is weaker than the usual notion of rigidity. That is, say A is an $n \times n$ matrix and A is (r, ns)-rigid then $r_A(r) \geq s$. From the perspective of rigidity upper bounds, if $r_A(r) \leq s$ then $R_A(r) \leq ns$. Thus, it follows that proving a family of matrices to be non-regular rigid is even a stronger criterion compared to the usual notion of non-rigidity. In this section, we show that the Vandermonde matrices are not Valiant rigid (refer subsection 1.3). Note that, for any matrix $M \in \mathbb{F}^{n \times n}$, showing that there exists constants c_1, c_2 and $\varepsilon > 0$ such that

$$R_M^{\mathbb{F}}\left(\frac{n}{\exp(\varepsilon^{c_1} \log^{c_2} n)}\right) \leq n^{1+\varepsilon}$$

implies that M is not Valiant rigid. For concreteness, we state our results in terms of $R_M(\cdot)$ and $r_M(\cdot)$.

We start by defining some interesting classes of matrices which will be used frequently in the subsequent sections.

Definition 9.3 (Discrete Fourier Transform (DFT) Matrices)**.** *Let $\mathbb{F}$ be any field and let $\omega \in \mathbb{F}$ be a primitive nth root of unity in $\mathbb{F}$. Then, the Discrete Fourier Transform (DFT) Matrix of order n, denoted by F_n is an $n \times n$ matrix over $\mathbb{F}$ defined as follows. The rows and columns of the matrix are indexed from $\{0, 1, \ldots, n-1\}$. The (i, j)th entry of F_n is ω^{ij}.* ⌟

Definition 9.4 (Circulant Matrices)**.** *Let $\mathbb{F}$ be a field and C_n be a matrix of order $n \times n$ with entries in $\mathbb{F}$ such that the rows and columns of C_n are indexed from $\{0, 1, \ldots, n-1\}$. Then, C_n is said to be a Circulant matrix if there exist $c_0, c_1, \ldots, c_{n-1} \in \mathbb{F}$ such that for every $i, j \in \{0, 1, \ldots, n-1\}$, $C_n(i, j) = c_{(i+j) \mod n}$.* ⌟

Definition 9.5. *Let $\mathbb{F}$ be a field. A Vandermonde matrix(V_n) of order $n \times n$ is defined by an input vector $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ where $\boldsymbol{\alpha} \in \mathbb{F}^n$. The rows and columns of V_n are indexed from $\{0, 1, \ldots, n-1\}$ and for every $i, j \in \{0, 1, \ldots, n-1\}$, $V_n(i, j) = \alpha_i^j$.* ⌟

We now state the formal version of Theorem 2.6.

Theorem 9.6. *Let $0 < \varepsilon < 0.01$, p be a fixed prime and c be a fixed constant. Let $a : \mathbb{N} \to \mathbb{N}$ be a function such that for all n, $a(n) \leq \log^c n$. Let $\{V_n\}_{n \in \mathbb{N}}$ be a family of matrices over $\overline{\mathbb{F}}_p$ such that for every n, V_n*

is an $n \times n$ Vandermonde Matrix with entries in the subfield $\mathbb{F}_{q=p^{a(n)}}$. Then, for all large enough n,

$$R_{V_n}^{\mathbb{F}_q}\left(\frac{n}{\exp(\Omega(\varepsilon^7 \log^{0.5} n))}\right) \leq n^{1+31\varepsilon}$$

As alluded in subsection 3.4, the main idea in proving Theorem 9.6 comes from viewing Algorithm 2 as a decomposition of (any) Vandermonde matrix into the product of a row-sparse matrix(A) and a sufficiently non-rigid matrix(B). Before getting into the proof of the main theorem, we discuss the non-rigidity of the matrix family "B". As it turns out, B is a multidimensional analog of the DFT matrix. Thus, we start by proving that the DFT matrices are non-rigid over quasi-polynomial size finite field $\mathbb{F}_q$ in Theorem 9.10. Followed by showing that the multidimensional DFT matrices are also non-rigid in Theorem 9.14, using the non-rigidity of DFT matrices and the Kronecker product structure of multidimensional DFT. Finally, in subsection 9.3, we flesh out the decomposition of Vandermonde matrices as product of row-sparse and multidimensional DFT matrices and use it to conclude that Vandermonde matrices are non-rigid as well.

9.1 Non-rigidity of DFT matrices

In this section, we give an upper bound on the rigidity of DFT matrices over finite fields. This bound follows directly from the work of Dvir and Liu [DL20] although the precise statement needed for our purpose (see Theorem 9.10) is not included in [DL20] as the results there are stated for a family of matrices over a fixed finite field. Whereas, in Theorem 9.6, the field is also increasing in size with the dimension of the matrix and for its proof, we need a bound on the rigidity of a family of DFT matrices with the field growing in size with the dimension of the matrices. This turns out to be a direct consequence of the results in [DL20] and was communicated to us by the Zeev Dvir and Allen Liu [DL21].

We start with a simple observation from [DL20].

Observation 9.7 ([DL20])**.** *Let M be an $n \times n$ matrix over a field $\mathbb{F}$, and let D be an $n \times n$ diagonal matrix over $\mathbb{F}$. Then, for every choice of rank parameter a,*

$$r_{DM}^{\mathbb{F}}(a) \leq r_M^{\mathbb{F}}(a)\,, \qquad \text{and} \qquad r_{MD}^{\mathbb{F}}(a) \leq r_M^{\mathbb{F}}(a)\,.$$

Let's analyse the DFT matrix F_{q-1} over $\mathbb{F}_q$. We show that there is a way to get a Circulant matrix by scaling F_{q-1} matrix over $\mathbb{F}_q$. This along with Observation 9.7 links the rigidity of F_{q-1} matrix over $\mathbb{F}_q$ with the rigidity of Circulant matrices.

Lemma 9.8. *It is possible to rescale the rows and columns of a DFT matrix F_{q-1} over any finite field $\mathbb{F}_q$ to get a Circulant matrix C_{q-1} over the extended field $\mathbb{F}_{q^2}$, where $q = p^a$.*

The proof of the above lemma is almost identical to the proof of Claim 2.22 in [DL20] and hence omitted. We would like to emphasise one key point though. The scaling procedure involves

working with an element ζ s.t. $\zeta^2 = g$, where g is the generator of $\mathbb{F}_q^*$. Note that ζ may not always exist in the underlying field $\mathbb{F}_q$, when q is odd. In that case, we perform a degree 2 extension over the base field $\mathbb{F}_q$. After extension, we view F_{q-1} over this extended field and perform appropriate scaling using ζ to get a Circulant matrix over $\mathbb{F}_{q^2}$.

We now state the upper bound on the rigidity of Circulant matrices from [DL20].

Theorem 9.9 (Theorem 7.27 in [DL20]). *Let $0 < \varepsilon < 0.01$ and p be a fixed prime. For all sufficiently large n, if C_n is an $n \times n$ Circulant matrix over $\mathbb{F}_p$ then,*

$$r_{C_n}^{\mathbb{F}_p}\Big(\frac{n}{\exp(\varepsilon^6(\log n)^{0.35})}\Big) \leq n^{15\varepsilon}$$

We now state an upper bound on the rigidity of DFT matrices due to Dvir & Liu which is the main takeaway of this subsection.

Theorem 9.10 ([DL20, DL21]). *Let $0 < \varepsilon < 0.01$, p be a fixed prime and c be a fixed constant. Let $a : \mathbb{N} \to \mathbb{N}$ be a function such that for all n, $a(n) \leq \log^c n$. Let $\{F_n\}_{n\in\mathbb{N}}$ be a family of matrices over $\overline{\mathbb{F}}_p$ such that for every n, F_n is an $n \times n$ DFT Matrix with entries in the subfield $\mathbb{F}_{p^{a(n)}} =: \mathbb{F}_q$. Then,*

$$r_{F_n}^{\mathbb{F}_q}\Big(\frac{n}{\exp(0.5 \cdot \varepsilon^6(\log n)^{0.35})}\Big) \leq 2\log_p q \cdot n^{15\varepsilon}$$

Proof. We first obtain a Circulant matrix C_n of order n by scaling the DFT matrix (of order n) from Lemma 9.8. As discussed earlier, the entries in the corresponding Circulant matrix potentially belongs to a degree 2 extension over the field $\mathbb{F}_q$, that is $C_n \in \mathbb{F}_{q^2}^{n\times n}$. Let $\mathbb{F}_{q^2}[X] = \mathbb{F}_p[X]/\langle v(X)\rangle$ where $v(X)$ is a degree $2a$ irreducible polynomial over $\mathbb{F}_p$. Hence, C_n can be expressed as $C_n = C^{(0)} + C^{(1)}X + C^{(2)}X^2 + \ldots + C^{(2a-1)}X^{2a-1}$, where each $C^{(i)}$ is also a Circulant matrix of order n over $\mathbb{F}_p$. This follows directly from the structure of Circulant matrices.

Now we invoke the rigidity upper bound from Theorem 9.9 for each of the above Circulant matrix $C^{(i)}$ over the fixed field $\mathbb{F}_p$. Due to the sub-additive property of rank and sparsity, the overall rank and sparsity of C_n gets upper bounded by $2a$ times the rank and sparsity of each of these $C^{(i)}$'s. Hence,

$$r_{F_n}^{\mathbb{F}_q}\Big(\frac{2an}{\exp(\varepsilon^6(\log n)^{0.35})}\Big) \leq 2an^{15\varepsilon}$$

On rewriting the above equation,

$$r_{F_n}^{\mathbb{F}_q}\Big(\frac{n}{\exp(\varepsilon^6(\log n)^{0.35} - \log\log_p q^2)}\Big) \leq 2\log_p q \cdot n^{15\varepsilon}$$

Since $a < \log^c n$, $\varepsilon^6 \log^{0.35} n \gg \log\log_p q^2$, this concludes the proof.

□

9.2 Non-rigidity of multidimensional DFT matrices

For our proof of Theorem 9.6, we need an upper bound on the rigidity of the following high dimensional analog of DFT Matrices.

Definition 9.11. *Let $m, d \in \mathbb{N}$, $\mathbb{F}$ be a field and $S \subset \mathbb{F}$ be a subset of cardinality d. The matrix $\mathcal{V}_{d,m}^{S}$ is a $d^m \times d^m$ matrix with rows labelled by all field elements of the product set $S^m = S \times S \times \cdots \times S$, and columns labelled by all monomials of individual degree at most $d-1$ in variables $\mathbf{x} = (x_1, x_2, \ldots, x_m)$. Such that, for $\mathbf{a} \in S^m$ and a monomial $\mathbf{x}^{\mathbf{e}}$ of individual degree at most $d-1$, the $(\mathbf{a}, \mathbf{e})$'th entry of $\mathcal{V}_{d,m}^{S}$ equals $\mathbf{a}^{\mathbf{e}}$.*

⌟

For most of this section, we work with $\mathcal{V}_{d,m}^{S}$ for the setting where $\mathbb{F}$ is a finite field of size q, $d = q$ and $S = \mathbb{F}$. Moreover, when S is clear from the context, we drop the superscript and denote $\mathcal{V}_{d,m}^{S}$ by $\mathcal{V}_{d,m}$.

Note that for $m = 1$ the matrix $\mathcal{V}_{q,1}^{\mathbb{F}_q}$ is closely related to the DFT matrix of order $q-1$ over $\mathbb{F}_q$. To see this, note that if g is a generator of the multiplicative group $\mathbb{F}_q^*$, then g is trivially a primitive root of unity of order $q-1$ over $\mathbb{F}_q$. So, up to a permutation of rows, the rows of $\mathcal{V}_{q,1}^{\mathbb{F}_q}$ can be viewed as being indexed by $g^0, g^1, g^2, \ldots, g^{q-1}$, and the columns by $0, 1, \ldots, q-1$ in this order. Thus, if we discard the row indexed by 0 and the column indexed by $q-1$ of $\mathcal{V}_{q,1}^{\mathbb{F}_q}$, what remains is precisely a DFT matrix of order $q-1$ over the field $\mathbb{F}_q$. The following lemma is an easy consequence of this observation together with the upper bound on rigidity of DFT matrices from [DL20].

Lemma 9.12. *Let $0 < \varepsilon < 0.01$, and $\mathbb{F}_q$ be a finite field of size q. Let $V = \mathcal{V}_{q,1}^{\mathbb{F}_q}$. Then, for all sufficiently large q,*

$$r_V^{\mathbb{F}_q}\left(\frac{q}{\exp(\varepsilon^6(\log q)^{0.34})}\right) \leq q^{16\varepsilon}$$

Proof. Consider the $(q-1) \times (q-1)$ submatrix $\bar{V}$ of V by deleting the row corresponding to the field element 0 and the column corresponding to the monomial of degree $q-1$. $\bar{V}$ is a DFT matrix of order $q-1$ with the generator of the multiplicative group $\mathbb{F}_q^*$ being the primitive root of unity of order $q-1$. Note that setting $n = q-1$ and invoking the rigidity upper bound of DFT matrices from Theorem 9.10, we get,

$$r_{\bar{V}}^{\mathbb{F}_q}\left(\frac{(q-1)}{\exp(0.5\varepsilon^6(\log(q-1))^{0.35})}\right) \leq 2\log_p q \cdot (q-1)^{15\varepsilon}.$$

Before simplifying the above expression, let's see how this related to rigidity of V. In order to do that, we re-add the deleted row and column to $\bar{V}$. This process can potentially increase the rank of the augmented matrix by at most 2. Also, note that $\log_p q \ll q^{\varepsilon}$, thus we get

$$r_V^{\mathbb{F}_q}\left(\frac{q}{\exp(\varepsilon^6((\log q)^{0.34})}\right) \leq q^{16\varepsilon}.$$

□

In the rest of this section, we generalize Lemma 9.12 to obtain an upper bound on the rigidity of $\mathcal{V}_{q,m}^{\mathbb{F}_q}$ for larger values of m. The following simple observation is the first step in this direction.

Observation 9.13. $\mathcal{V}_{q,m}^{\mathbb{F}_q} =^{10} \left(\mathcal{V}_{q,1}^{\mathbb{F}_q}\right)^{\otimes m}$.

We are now ready to state the main theorem of this section, where we prove a non-trivial upper bound on the rigidity of $\mathcal{V}_{q,m}^{\mathbb{F}_q}$ for an appropriate range of parameters.

Theorem 9.14. *Let $0 < \varepsilon < 0.01$, and $m \in \mathbb{N}^+$ be the given parameters and $\mathbb{F}_q$ be any finite field such that $m \leq q^{\varepsilon}$. Then for large enough q , we have*

$$r_{\mathcal{V}_{q,m}}^{\mathbb{F}_q}\left(\frac{q^m}{\exp\left(9\varepsilon^7(\log q)^{0.34}m\right)}\right) \leq q^{27\varepsilon \cdot m}$$

Proof. From Observation 9.13, we have

$$\mathcal{V}_{q,m}^{\mathbb{F}_q} = \left(\mathcal{V}_{q,1}^{\mathbb{F}_q}\right)^{\otimes m}.$$

From Lemma 9.12, we have that

$$r_{\mathcal{V}_{q,1}}^{\mathbb{F}_q}\left(\frac{q}{\exp(\varepsilon^6(\log q)^{0.34})}\right) \leq q^{16\varepsilon}.$$

Thus, $\mathcal{V}_{q,1}$ can be written as the sum of a matrix L of rank at most $\frac{q}{\exp(\varepsilon^6(\log q)^{0.34})}$ and a matrix S with row sparsity at most $q^{16\varepsilon}$. Thus, $\mathcal{V}_{q,m}^{\mathbb{F}_q}$ can be written as

$$\mathcal{V}_{q,m}^{\mathbb{F}_q} = (L_1 + S_1) \otimes (L_2 + S_2) \otimes \ldots \otimes (L_m + S_m),$$

where each L_i is a copy of L and each S_i is a copy of S (they have been indexed for clarity of notation).

The above Kronecker product has 2^m many terms, each term consisting of the Kronecker products of various copies of L_i and S_j. We partition these summands into two groups based on the number of copies of L participating in the Kronecker product. To complete the proof, we show that the sum of every term with many copies of L is a matrix of *not too high* rank and the sum of the remaining terms (that have few copies of L and hence many copies of S) are non-trivially

[10] This equality holds up to some row column permutation, which doesn't affect rigidity.

sparse. This would complete the proof of the theorem. We now fill in the details, which involve some slightly careful calculations to get the quantitative bounds stated in the theorem.

We pick a threshold t (to be set later) and collect all terms in the Kronecker product

$$\mathcal{V}_{q,m}^{\mathbb{F}_q} = (L_1 + S_1) \otimes (L_2 + S_2) \otimes \ldots \otimes (L_m + S_m)\,,$$

consisting of at most t copies of S for $t = m(1 - 10\varepsilon)$. Let $\mathcal{L}$ be the sum of all such terms, and let $\mathcal{S}$ be the sum of the remaining terms. In the following two claims, we obtain an upper bound on the rank of $\mathcal{L}$ and the sparsity of $\mathcal{S}$.

Claim 9.15.

$$\operatorname{rank}(\mathcal{L}) \leq \frac{q^m}{\exp(9\varepsilon^7(m \cdot (\log q)^{0.34}))}\,.$$

Claim 9.16. *Every row and column of $\mathcal{S}$ has at most $q^{27\varepsilon \cdot m}$ non-zero entries.*

These bounds, together with the decomposition

$$\mathcal{V}_{q,m}^{\mathbb{F}_q} = \mathcal{L} + \mathcal{S}$$

complete the proof of the theorem. □

We now prove Claim 9.15 and Claim 9.16.

Proof of Claim 9.15. From Lemma 9.12, we know the rank of each L_i is at most $\left(\frac{q}{\exp\left(\varepsilon^6(\log q)^{0.34}\right)}\right)$ and the rank of each S_i is at most q since it is a $q \times q$ matrix. Moreover, the rank of a Kronecker product of matrices is the product of their ranks. Thus, we have

$$\begin{aligned}
\operatorname{rank}(\mathcal{L}) &\leq \sum_{j=0}^{t-1} \binom{m}{j} \left(\frac{q}{2^\alpha}\right)^{m-j} q^j && \text{(where } \alpha = \varepsilon^6(\log q)^{0.34}\text{)} \\
&= \left(\frac{q}{2^\alpha}\right)^m \sum_{j=0}^{t-1} \binom{m}{j} 2^{\alpha j} \\
&\leq \left(\frac{q}{2^\alpha}\right)^m 2^{\alpha t} \sum_{j=0}^{t-1} \binom{m}{j} \\
&\leq \left(\frac{q}{2^\alpha}\right)^m 2^{\alpha t} \cdot 2^m \\
&= (q)^m 2^{\alpha(t-m)+m}\,.
\end{aligned}$$

We now obtain an upper bound on the quantity $2^{\alpha(t-m)+m}$ to complete the proof of the claim.

$$2^m 2^{\alpha(t-m)} \leq 2^{(-10\varepsilon\alpha+1)m} \qquad \text{(since } t = m(1-10\varepsilon)\text{)}$$

$$\leq 2^{-\alpha \cdot 9\varepsilon m} \qquad \text{since } \alpha \cdot 9\varepsilon > 1 \text{ for all large enough } q$$

Thus, the rank of $\mathcal{L}$ is at most $q^m \cdot 2^{-9\varepsilon\alpha m}$, which by the choice of α gives

$$\text{rank}(\mathcal{L}) \leq \frac{q^m}{\exp(9\varepsilon^7 m \cdot (\log q)^{0.34})}\,.$$

□

Proof of Claim 9.16. This proof also proceeds along the lines of the proof of Claim 9.15, and involves similar calculations. We just note that every summand in $\mathcal{S}$ involves at least $(m-t)$ sparse matrices in the Kronecker product. Moreover, we rely on the basic fact that the row/column sparsity of a Kronecker product of matrices is equal to the product of the row/column sparsity of each of the matrices in the product.

Let $q' = q^{16\varepsilon}$ denote the row sparsity of every matrix S_i. For each L_i, we use the obvious upper bound of q on its row sparsity for our estimate.

$$\begin{aligned}
\mathsf{RowSparsity}(\mathcal{S}) &\leq \sum_{j=t}^{m} \binom{m}{j} (q')^j (q)^{m-j} \\
&= q^m \cdot \sum_{j=t}^{m} \binom{m}{j} \left(\frac{q'}{q}\right)^j \\
&\leq q^m \sum_{j=t}^{m} m^j \left(\frac{q'}{q}\right)^j \\
&\leq 2.q^m \left(\frac{q'm}{q}\right)^t && \text{(since } q'm/q < 1/2\text{)}\,. \\
&\leq q^m \left(\frac{1}{q^{1-16\varepsilon-\log_q m}}\right)^t \\
&\leq q^m \left(\frac{1}{q^{1-17\varepsilon}}\right)^t && \text{(we have } \frac{\log m}{\log q} < \varepsilon\text{)}
\end{aligned}$$

Using $t = m(1-10\varepsilon)$, we get

$$\mathsf{RowSparsity}(\mathcal{S}) \leq q^{m(1-(1-10\varepsilon)(1-17\varepsilon))} \leq q^{(10\varepsilon+17\varepsilon)m} \leq q^{27\varepsilon \cdot m}\,.$$

An almost identical argument also bounds the column sparsity, which completes the proof of the claim.

□

9.3 Non-rigidity of Vandermonde matrices

In this section, we prove Theorem 9.6. As discussed in subsection 3.4, using the algorithm for multivariate multipoint evaluation, we write (any) Vandermonde matrix as the product of a sparse matrix and a multidimensional DFT matrix. Then, we invoke the rigidity upper bound for multidimensional DFT matrices in Theorem 9.14. This together with the fact that a product of sparse and non-rigid matrix continues to be non-rigid with slightly diminished parameters completes the proof. We now fill in the details.

Proof of Theorem 9.6. We start by setting up some necessary notation. Let d, m be parameters, chosen as follows.

- $m := \left\lceil \log^{0.3} n \right\rceil$
- d is set to be the smallest integer such that $d^m > n$

Clearly, d can be taken to be at most $n^{1/m} + 1$. For this choice of d, m, we note that d^m is at least n and most $(n^{1/m} + 1)^m \leq 2n$.

Let $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ be the generators of the Vandermonde matrix V_n , i.e., $V_n(i,j) = \alpha_i^j$. For every $i \in \{0, 1, \ldots, n-1\}$, let $\boldsymbol{\alpha}_i = (\alpha_i, \alpha_i^d, \ldots, \alpha_i^{d^{m-1}})$. For simplicity, we use a to denote $a(n)$. Let b be the smallest integer such that $padm \geq q_0 = p^b > adm$. Let W be the $p^{bm} \times p^{bm}$ matrix with its rows indexed by vectors in $\mathbb{F}_{p^b}^m$ and the columns indexed by all m-variate monomials of individual degree at most $q_0 - 1$ and the $(\mathbf{c}, \mathbf{e})$ entry of W is equal to $\mathbf{c}^{\mathbf{e}}$. Moreover, let $\tilde{V}$ be the matrix with rows indexed by $\{0, 1, \ldots, n-1\}$ and columns indexed by all m-variate monomials of individual degree $d - 1$ and the $(i, \mathbf{e})$ entry being equal to $\boldsymbol{\alpha}_i^{\mathbf{e}}$.

To see the connection between the matrices V_n, $\tilde{V}$ and W, let us try to understand the action of V_n on a vector. Semantically, we can view this n dimensional vector as the coefficient vector of a univariate polynomial f of degree at most $n - 1$ and thus the matrix vector product $V_n \cdot \mathrm{Coeff}(f)$ is precisely the evaluation vector of f on inputs $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$. Recall the inverse Kronecker map $\psi_{d,m}$ from Definition 8.2, and let F be an m-variate polynomial of degree at most $d - 1$ in each variable such that $F = \psi_{d,m}(f)$. For this to make sense, recall that by our choice of parameters $d^m \geq n$. Now, it follows from these definitions that for every $i \in \{0, 1, \ldots, n-1\}$, $f(\alpha_i) = F(\boldsymbol{\alpha}_i)$. Moreover, the coefficient vectors of f and F are closely related. In fact, if $d^m = n$, then these coefficient vectors are exactly the same (even though the natural labelling of the coordinates of $\mathrm{Coeff}(f)$ is via univariate monomials of degree at most $n - 1$ and that of $\mathrm{Coeff}(F)$ is via m-variate monomials of individual degree at most $d - 1$). If $d^m > n$, then we have to append some zeroes to the coefficient vector f to obtain the coefficient vector of F. In other words,

$$\mathrm{Coeff}(F) = \tilde{I} \cdot \mathrm{Coeff}(f)\,,$$

where $\tilde{I}$ is a $d^m \times n$ matrix with the top $n \times n$ submatrix being the identity matrix and the remaining $d^m - n$ rows being all zeroes. Moreover,

$$\tilde{V} \cdot \mathrm{Coeff}(F) = V_n \cdot \mathrm{Coeff}(f)\,.$$

Now we recall Algorithm 2 for multivariate multipoint evaluation and invoke it for the polynomial F and evaluation points $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{n-1}$. The algorithm first evaluates F on the product set $\mathbb{F}_{q_0}^m$ in the preprocessing phase. In other words, it computes the vector $W \cdot \mathrm{Coeff}(F)$. Then, for every $i \in \{0, 1, \ldots, n-1\}$, the value of F on $\boldsymbol{\alpha}_i$ is computed by some *local* computation consisting of looking at the univariate polynomial $h_i(t)$ obtained as a restriction of F on a curve of degree at most $a-1$ through $\boldsymbol{\alpha}_i$, then interpolating h_i using the already available values of F on $\mathbb{F}_{q_0}^m$, and then evaluating h_i on an appropriate input to get $F(\boldsymbol{\alpha}_i)$. In other words, $F(\boldsymbol{\alpha}_i)$ is obtained by taking an appropriate weighted linear combination of the value of F on a specific subset of points in $\mathbb{F}_{q_0}^m$ of size at most adm. Thus, for every $i \in \{0, 1, \ldots, n-1\}$, there is a vector τ_i of length q_0^m with entries in the field $\mathbb{F}_{p^{ab}}$[11] such that

$$F(\boldsymbol{\alpha}_i) = \langle \tau_i, W \cdot \mathrm{Coeff}(F) \rangle\,.$$

If we collect the vectors $\tau_0, \tau_1, \ldots, \tau_{n-1}$ into an $n \times q_0^m$ matrix Γ, then we have

$$\tilde{V} \cdot \mathrm{Coeff}(F) = \Gamma \cdot W \cdot \mathrm{Coeff}(F)\,.$$

Recalling the relation between F, f and between V_n, $\tilde{V}$, we get that

$$V_n \cdot \mathrm{Coeff}(f) = \tilde{V} \cdot \mathrm{Coeff}(F) = \Gamma \cdot W \cdot \mathrm{Coeff}(F)\,,$$

or, using $\mathrm{Coeff}(F) = \tilde{I} \cdot \mathrm{Coeff}(f)$, we get

$$V_n \cdot \mathrm{Coeff}(f) = \left(\Gamma \cdot W \cdot \tilde{I}\right) \cdot \mathrm{Coeff}(f)\,.$$

Recall that we started with $\mathrm{Coeff}(f)$ being an arbitrary vector. Thus, we have

$$V_n = \Gamma \cdot W \cdot \tilde{I}\,.$$

Now, to obtain a decomposition of V_n as the sum of a sparse and a low rank matrix, the idea is to invoke Theorem 9.14 on W, and combine the decomposition obtained together with the sparsity of Γ and $\tilde{I}$ to obtain a similar decomposition for V_n. To invoke Theorem 9.14, we must satisfy all the constraints on the parameters present there. To this end, we note that

[11] Recall that for technical reasons, we have to work in the field $\mathbb{F}_{p^{ab}}$ in Algorithm 2 even though the inputs are in $\mathbb{F}_{p^a}$.

- $q_0 > adm$ and thus is large enough.
- $\log m / \log q_0 < \varepsilon$. This is true since $\log m / \log q_0 < \log m / \log d < \frac{0.3 \log\log n}{\log^{0.7} n} \ll \varepsilon$.

Now from Theorem 9.14, we have that W can be written as the sum of matrices L and S where the row and column sparsity of S is at most $q_0^{27\varepsilon m}$ and the rank of L can be upper bounded as follows.

$$\text{rank}(L) \leq \frac{q_0^m}{\exp(9\varepsilon^7 (\log q_0)^{0.34} m)} \leq \frac{(padm)^m}{\exp(9\varepsilon^7 (\log q_0)^{0.34} m)}\,.$$

Note that by our choice of parameters, $(pam)^m \ll 2^{\log^{0.4} n}$ and $m \log^{0.34} q_0 \geq (\log n)^{(0.34)(0.70)+0.3} \geq \log^{(0.54)} n$. Also, since $2^{(\varepsilon^7 m \log^{0.34} q_0)} \gg (pam)^m$, we get

$$\text{rank}(L) \leq \frac{q_0^m}{\exp(9\varepsilon^7 (\log q_0)^{0.34} m)} \leq \frac{n}{\exp(\Omega(\varepsilon^7 \log^{0.5} n))}\,.$$

Now, we have

$$V_n = \Gamma \cdot W \cdot \tilde{I} = \Gamma \cdot (L + S) \cdot \tilde{I}\,,$$

or after simplification,

$$V_n = (\Gamma \cdot L \cdot \tilde{I}) + (\Gamma \cdot S \cdot \tilde{I})\,.$$

Clearly, the rank of $(\Gamma \cdot L \cdot \tilde{I})$ is at most the rank of L which as we calculated above is at most $\frac{n}{\exp(\Omega(\varepsilon^7 \log^{0.5} n))}$. The row sparsity of $(\Gamma \cdot S \cdot \tilde{I})$ is at most the product of the row sparsity of Γ and the row sparsity of S [12] and thus is at most

$$\begin{aligned}(adm)(q_0^{27\varepsilon \cdot m}) &\leq (padm)((q_0)^m)^{27\varepsilon} \\ &= p^{27\varepsilon \cdot m+1} \cdot a^{27\varepsilon \cdot m+1} n^{27\varepsilon} m^{(27\varepsilon \cdot m+1)} d\end{aligned}$$

Note that by our choice of parameters, $d < n^\varepsilon$, $a^{27\varepsilon \cdot m+1} < n^\varepsilon$, $p^{27\varepsilon \cdot m+1} < n^\varepsilon$ and $m^{(27\varepsilon \cdot m+1)} < n^\varepsilon$, and thus, the row sparsity of $(\Gamma \cdot S \cdot \tilde{I})$ is at most $n^{31\varepsilon}$.

Thus, V_n can be written as the sum of a matrix of row sparsity[13] at most $n^{31\varepsilon}$ and rank at most $\frac{n}{\exp(\Omega(\varepsilon^7 \log^{0.5} n))}$ as claimed in the theorem.

□

[12] $\tilde{I}$ has row and column sparsity 1, so it does not affect the calculations.

[13] We emphasize the point that we only deal with row sparsity here and *not both row and column sparsity*.

Acknowledgements

Mrinal thanks Swastik Kopparty for introducing him to the work of Kedlaya & Umans [KU11] and the question of multipoint evaluation and numerous invaluable discussions.

We thank Zeev Dvir and Allen Liu for answering our questions regarding the results in [DL20] and for allowing us to include the proof sketch of Theorem 9.10 in this paper. We also thank Ben Lund for helpful discussions and references on the finite fields Kakeya problem and for pointing us to the relevant literature on Furstenberg sets, both of which indirectly played a role in some of the ideas in this paper.

Finally, we thank Prerona Chatterjee, Prahladh Harsha, Ramprasad Saptharishi and Aparna Shankar for sitting through a (pretty sketchy) presentation of earlier versions of some of the proofs in this paper and for much encouragement.

References

[AC19] JOSH ALMAN and LIJIE CHEN. *Efficient construction of rigid matrices using an NP oracle*. In *Proc. 60th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 1034–1055. 2019. doi:10.1109/FOCS.2019.00067. 5

[Alm21] JOSH ALMAN. *Kronecker products, low-depth circuits, and matrix rigidity*. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 772–785. ACM, 2021. doi:10.1145/3406325.3451008. 5

[AW17] JOSH ALMAN and R. RYAN WILLIAMS. *Probabilistic rank and matrix rigidity*. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, pages 641–652. 2017. arXiv:1611.05558, doi:10.1145/3055399.3055484. 5

[Ber70] E. R. BERLEKAMP. *Factoring polynomials over large finite fields*. Mathematics of Computation, 24(111):713–735, 1970. 19

[BHPT20] AMEY BHANGALE, PRAHLADH HARSHA, ORR PARADISE, and AVISHAY TAL. *Rigid matrices from rectangular PCPs or Hard Claims have Complex Proofs*. In *Proc. 61st IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 858–869. 2020. arXiv:2005.03123, eccc:2020/TR20-075, doi:10.1109/FOCS46700.2020.00084. 5

[BKW19] ANDREAS BJÖRKLUND, PETTERI KASKI, and RYAN WILLIAMS. *Generalized kakeya sets for polynomial evaluation and faster computation of fermionants*. Algorithmica, 81(10):4010–4028, 2019. doi:10.1007/s00453-018-0513-7. 3, 4, 8, 9, 14, 15

[BM74] A. BORODIN and R. MOENCK. *Fast modular transforms*. Journal of Computer and System Sciences, 8(3):366–386, 1974. doi:https://doi.org/10.1016/S0022-0000(74)80029-2. , 1, 2

[DE19] ZEEV DVIR and BENJAMIN L. EDELMAN. *Matrix rigidity and the Croot-Lev-Pach lemma*. Theory Comput., 15(8):1–7, 2019. arXiv:1708.01646, doi:10.4086/toc.2019.v015a008. 5

[DL20] Zeev Dvir and Allen Liu. *Fourier and circulant matrices are not rigid*. Theory of Computing, 16(20):1–48, 2020. doi:10.4086/toc.2020.v016a020. , 5, 10, 16, 43, 44, 45, 52

[DL21] ———. personal communication, 2021. 43, 44

[For14] Michael Forbes. *Polynomial Identity Testing of Read-Once Oblivious Algebraic Branching Programs*. Ph.D. thesis, Massachusetts Institute of Technology, 2014. 20

[Fri93] Joel Friedman. *A note on matrix rigidity*. Combinatorica, 13(2):235–239, 1993. doi:10.1007/BF01303207. 5

[GG03] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003. 22

[GT18] Oded Goldreich and Avishay Tal. *Matrix rigidity of random Toeplitz matrices*. Comput. Complexity, 27(2):305–350, 2018. Preliminary version in *48th STOC*, 2002). eccc:2015/TR15-079, doi:10.1007/s00037-016-0144-9. 5

[Juk10] Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer Publishing Company, Incorporated, 1st edition, 2010. 17

[Kiv21] Bohdan Kivva. *Improved Upper Bounds for the Rigidity of Kronecker Products*. In Filippo Bonchi and Simon J. Puglisi, eds., *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021. doi:10.4230/LIPIcs.MFCS.2021.68. 5

[KU11] Kiran S. Kedlaya and Christopher Umans. *Fast polynomial factorization and modular composition*. SIAM J. Comput., 40(6):1767–1802, 2011. doi:10.1137/08073408X. , 1, 2, 3, 4, 5, 7, 8, 9, 22, 40, 52

[Lok00] Satyanarayana V. Lokam. *On the rigidity of vandermonde matrices*. Theor. Comput. Sci., 237(1-2):477–483, 2000. doi:10.1016/S0304-3975(00)00008-6. 5, 6

[Lok01] ———. *Spectral methods for matrix rigidity with applications to size-depth trade-offs and communication complexity*. J. Comput. Syst. Sci., 63(3):449–473, 2001. doi:10.1006/jcss.2001.1786. 5

[Lok06] ———. *Quadratic lower bounds on matrix rigidity*. In *Theory and Applications of Models of Computation, Third International Conference, TAMC 2006, Beijing, China, May 15-20, 2006, Proceedings*, pages 295–307. 2006. doi:10.1007/11750321_28. 5

[Mil95] Peter Bro Miltersen. *On the cell probe complexity of polynomial evaluation*. Theor. Comput. Sci., 143(1):167–174, May 1995. doi:10.1016/0304-3975(95)80032-5. , 4, 5, 9, 40

[NZ04] Michael Nüsken and Martin Ziegler. *Fast multipoint evaluation of bivariate polynomials*. In Susanne Albers and Tomasz Radzik, eds., *Algorithms – ESA 2004*, pages 544–555. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 2

[Sho90] Victor Shoup. *New algorithms for findings irreducible polynomials over finite fields*. Mathematics of Computation, 54(189):435–447, 1990. 18, 19

[SSS97] MOHAMMAD AMIN SHOKROLLAHI, DANIEL A. SPIELMAN, and VOLKER STEMANN. *A remark on matrix rigidity*. Inform. Process. Lett., 64(6):283–285, 1997. doi:10.1016/S0020-0190(97)00190-7. 5

[Uma08] CHRISTOPHER UMANS. *Fast polynomial factorization and modular composition in small characteristic*. In CYNTHIA DWORK, ed., *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 481–490. ACM, 2008. doi:10.1145/1374376.1374445. , 1, 2, 3, 4, 7, 8, 9

[Val77] LESLIE G. VALIANT. *Graph-theoretic arguments in low-level complexity*. In JOZEF GRUSKA, ed., *Proc. 6th Symposium of Mathematical Foundations of Computer Science (MFCS)*, volume 53 of *LNCS*, pages 162–176. Springer, 1977. doi:10.1007/3-540-08353-7_135. , 5, 9, 16, 41

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il