FLI: Folding Lookup Instances

Albert Garreta[1] and Ignacio Manzur[1]

[1]Nethermind Research, {albert,ignacio}@nethermind.io

September 30, 2024

Abstract

We introduce two folding schemes for lookup instances: FLI and FLI+SOS. Both use a PIOP to check that a matrix has elementary basis vectors as rows, with FLI+SOS adding a twist based on Lasso's [STW24] SOS-decomposability.

FLI takes two lookup instances $\{\mathbf{a}_1\}, \{\mathbf{a}_2\} \subseteq \{\mathbf{t}\}$, and expresses them as matrix equations $M_i \cdot \mathbf{t}^\mathsf{T} = \mathbf{a}_i^\mathsf{T}$ for $i = 1, 2$, where each matrix $M_i \in \mathbb{F}^{m \times N}$ has rows which are elementary basis vectors in $\mathbb{F}^N$. Matrices that satisfy this condition are said to be in $\mathbf{R}_{\mathsf{elem}}$. Then, a folding scheme for $\mathbf{R}_{\mathsf{elem}}$ into a relaxed relation is used, which combines the matrices M_1, M_2 as $M_1 + \alpha M_2$ for a random $\alpha \in \mathbb{F}$. Finally, the lookup equations are combined as $(M_1 + \alpha M_2) \cdot \mathbf{t}^\mathsf{T} = (\mathbf{a}_1 + \alpha \mathbf{a}_2)^\mathsf{T}$. In FLI, only the property that a matrix is in $\mathbf{R}_{\mathsf{elem}}$ is folded, and this makes the FLI folding step the cheapest among existing solutions. The price to pay is in the cost for proving accumulated instances.

FLI+SOS builds upon FLI to enable folding of large SOS-decomposable [STW24] tables. This is achieved through a variation of Lasso's approach to SOS-decomposability, which fits FLI naturally. For comparison, we describe (for the first time to our knowledge) straightforward variations of Protostar [BC23] and Proofs for Deep Thought [BC24] that also benefit from SOS-decomposability. We see that for many reasonable parameter choices, and especially those arising from lookup-based zkVMs [AST24], FLI+SOS can concretely be the cheapest folding solution.

Contents

1 Introduction

Folding schemes have been an active area of research in the last few years [BGH19, BCMS20, BCL+21, KST22, KS24, BC23, EG23, BC24]. Informally, these schemes can be described as interactive proofs in which a Prover and a Verifier create a new instance-witness pair for a certain relation $\mathbf{R}_2$ from two instances-witness pairs for relations $\mathbf{R}_1, \mathbf{R}_2$. The validity of the newly created instance-witness pair implies the validity of the two original instance-witness pairs. The idea is that if this combination process is less expensive than directly proving that the two instance-witness pairs belong to the relevant relations (in Prover time, memory requirements, or proof size), one can save on costs by reducing the task of proving that many instance-witness pairs belong to $\mathbf{R}_1$ to proving that a single pair belongs to $\mathbf{R}_2$. Initially, these schemes were created with the intention of improving the construction of primitives like Incrementally Verifiable Computation (IVC) [Val08] and Proof-Carrying-Data (PCD) [CT10].

Another active area of research is that of lookup arguments: these are arguments that allow a Prover to convince a Verifier that all elements in a vector $\mathbf{a}$ appear in a pre-established vector $\mathbf{t}$. The vectors $\mathbf{t}$ is often referred to as a *lookup table*, and we use the phrase "look $\mathbf{a}$ into $\mathbf{t}$" to mean engaging in the lookup argument to convince a Verifier that all elements in $\mathbf{a}$ appear in $\mathbf{t}$. Lookup arguments have become very popular within the context of SNARKs [Kil92, Mic94], because they allow for an efficient treatment of operations that are otherwise difficult to arithmetize. In SNARKs, these operations could be non-native field arithmetic, elliptic curve operations, binary operations, and so on. With lookup arguments, the Prover can use a lookup into a lookup table for the corresponding operation (this table is usually pre-defined), instead of needing to represent the computation in the arithmetization of the SNARK. Recent work in the field of lookup arguments can be essentially split between lookup arguments that use matrix equations [ZBK+22, ZGK+22, STW24] and those that use logarithmic derivatives [Hab22, EFG22, PH23]. Lasso [STW24] is the state-of-the-art matrix-based lookup argument without pre-processing, provided that the lookup table $\mathbf{t}$ has a specific structure, called *SOS-decomposability*. This informally says that to find an entry of the lookup table $\mathbf{t}$, one can evaluate a multilinear polynomial g at the entries of smaller tables $\mathbf{t}_1, \ldots, \mathbf{t}_\alpha$ in some structured way (for details, see Section 3.4). The typical example is a range-check table. For instance, to show that a field element is smaller than 2^{256}, one can break the element into 64 (or 32, or 16, and so on) bit parts, and perform a range check for each of these parts. Note that it is not even possible to materialize the table of all elements smaller than 2^{256}. Hence SOS-decomposability gives the ability to perform lookups into gigantic tables.

In a companion paper to Lasso, called Jolt [AST24], the authors construct a lookup-based zero-knowledge Virtual Machine (zkVM). This realizes an idea sketched out in a ZKResearch blogpost [bar22] called the *lookup singularity*, which pushes to the extreme the idea of using lookup arguments for verifying computation: instead of only verifying those

computations which are expensive or difficult to arithmetize with lookups, all operations are verified with lookups. In [AST24] the authors show that almost all operations of the RISC-V ISA are SOS-decomposable, and so using Lasso they are able to construct a lookup-based zkVM for the RISC-V instruction set. In a follow-up blogpost and talk [Tha24, AZ24], the authors express that one of the main steps in Jolt's roadmap is to implement *continuations*. This means breaking the CPU execution into chunks, and aggregating the proof of correctness of each chunk. The motivation for doing so is to reduce the peak memory consumption of generating a proof, but it comes at the cost of increasing the proof size (since now there is a proof for each chunk). This also leads to the need for proving many polynomial evaluations (several per chunk). As outlined in [Tha24], the authors plan to avoid these problems in two ways (each with its own use cases): one by using recursion and the polynomial commitment scheme from Binius [DP24, DP23], and the second by using folding schemes to aggregate the chunks.

1.1 Our contributions

In this paper, inspired by the use case of continuations in lookup-based zkVMs, we develop two folding schemes for lookup instances (see Definition 3.5) called FLI and FLI+SOS. The latter is an extension of FLI that is capable of leveraging SOS-decomposability of the lookup table $\mathbf{t}$. We use two main technical ingredients: a variation of the way Lasso uses SOS decompositions, and a PIOP and a folding scheme for the relation that a matrix has elementary basis vectors as rows (i.e. each row consists entirely of 0, except for one entry being 1).

Say we want to prove that the elements in $\mathbf{a} \in \mathbb{F}^m$ appear in $\mathbf{t} \in \mathbb{F}^N$. We refer to $\mathbf{a}$ as the small table, and to $\mathbf{t}$ as the big table. There are two types of costs that we focus on: the folding costs, and the cost of proving accumulated instances. The former is the Prover and Verifier cost (in field/group operations, random oracle costs, and so on) associated with the folding scheme, and the latter is the Prover and Verifier cost of a SNARK for the accumulated relation.

Irrespective of whether $\mathbf{t}$ is SOS-decomposable (Definition 3.6) or not, FLI has the cheapest folding Prover and Verifier (among the schemes described in Section 1.2), see Tables 1 to 3 and Section 6. The Prover folding costs of FLI are linear on m, i.e. the size of the small table $\mathbf{a}$. However, as is the case with the rest of analysed schemes, proving an accumulated instance requires incurring a cost of $O(N)$ at least (recall N is the size of $\mathbf{t}$). Hence, FLI, as well as the rest of folding schemes for lookup instances, cannot reasonably handle gigantic tables $\mathbf{t}$.

With this in mind we consider the case when $\mathbf{t}$ is SOS-decomposable, and design a variation of FLI, called FLI+SOS, which leverages SOS-decomposability of $\mathbf{t}$ in a natural way. This enables folding lookup instances where $\mathbf{t}$ is gigantic but SOS-decomposable, without incurring $O(N)$ costs, neither in the folding step, nor when proving accumulated instances. To

compare FLI+SOS, we describe straightforward variations of Protostar [BC23] and Proofs for Deep Thought (abbreviated DT) [BC24] that make use of the SOS decomposability of $\mathbf{t}$. See Section 1.2 for more details. We emphasize that as is, neither Protostar nor DT support SOS decompositions. To our knowledge, we are the first to describe the variations of these schemes that are compatible with SOS decompositions, and we make a number of favorable assumptions regarding their costs when comparing them to FLI+SOS. We call these variations Protostar+SOS and DT+SOS.

When $\mathbf{t}$ is SOS-decomposable into $\alpha = k \cdot c$ tables of size $N^{1/c}$, we show in Section 6 that for choices of m, N, c, k that naturally arise in the context of lookup-based zkVMs, FLI+SOS can overall be the cheapest folding scheme for lookup instances. Therefore it is a candidate for implementing continuations by folding computation chunks. For instance, we show that for $m = 2^{17}, N = 2^{1024}, c \sim 256$, $\alpha = 2c$ (this particular choice seems to be perfectly plausible in practice when using Jolt [AST24, Tha24]), with $n_f = 2^3$ foldings then:

- FLI+SOS's folding Prover is more than $4\times$ cheaper than Protostar+SOS's, and is cheaper than DT+SOS's (see Remark 6.1). In general if $\alpha = k \cdot c$, then FLI+SOS's folding Prover is more than $2 \cdot k\times$ cheaper than Protostar+SOS's.

- FLI+SOS's folding Verifier is comparable to (but slightly cheaper than) Protostar+SOS's Verifier, but much cheaper than DT+SOS's.

- FLI+SOS's folding Verifier has the lowest random oracle query costs. This is particularly relevant in the context of IVC [Val08], where the folding Verifier should be represented recursively in a circuit. Each random oracle query could potentially represent numerous and complicated constraints, as some hash functions that heuristically instantiate the random oracle are difficult to arithmetize.

We describe a custom SNARK for the accumulated/relaxed lookup relation in FLI+SOS, namely $\mathbf{R}^{\mathsf{accSOS}}$ (see Section 5.4). In our comparison, still with the same parameter values for m, N, c and k, we find that:

- Putting opening proofs for multilinear polynomials to the side, the Prover for accumulated instances of FLI+SOS is around 1.2 times more expensive than that of Protostar+SOS and around $1.3\times$ more expensive than that of DT+SOS. However, in this regime, DT+SOS's folding Prover is prohibitively expensive. *We emphasize that this result is obtained while making a number of optimistic simplifications regarding the cost of proving accumulated instances with Protostar+SOS and DT+SOS.* The improvement could be sharper, cf. Section 6 and in particular Remark 6.2.

- The polynomial opening proof cost of FLI+SOS is around $2\times$ less than Protostar+SOS and DT+SOS's optimistic cost for proving accumulated instances. We emphasize that this estimate assumes a naive curve-based (MSM) commitment scheme such as

PST [PST13, CGG+23] (i.e. a "multilinear KZG"), and that one can choose alternative schemes which removes this overhead, at the expense of increasing the folding Verifier work. For example, #2 in [Tha24] proposes using a tensor-like variation of Zeromorph[KT23] or HyperKZG[Set24] which would make this step no longer be the bottleneck in FLI+SOS, and would increase the Verifier work by $O(N^{1/c})$ group operations. With this, FLI+SOS's Verifier would be comparable to DT+SOS's and more expensive than Protostar+SOS's. However, FLI+SOS would be the cheapest scheme both in terms of the folding Prover work and the cost of proving accumulated instances.

For more details about the comparison, see Section 6 and Tables 1 to 4.

Potential usability in lattices. Finally, we remark that FLI is based solely on the sumcheck protocol, and because of that, it could potentially be used in the context of lattice-based cryptography. This is in contrast to Protostar and DT, which rely on field-based identities involving logarithmic derivatives [Hab22, PH23] which do not seem to carry over to lattices.

1.2 Related work

To the best of our knowledge, there are three available approaches to folding lookup instances. Hypernova [KS24] describes one such scheme in which the Prover cost is $O(N)$, while the Verifier's work is $O(m \log(N))$, where here N is the size of the big table $\mathbf{t}$, and m is the size of the small table $\mathbf{a}$. Protostar [BC23] describes a folding scheme based on the logUp lookup argument [Hab22] with Prover's costs $O(m)$, with the concrete costs being rather large due to the need of committing to 2 size-m vectors with entries of arbitrary length. A related posterior work, Proofs for Deep Thought (DT in short) [BC24], presents an alternative folding scheme for lookup instances in which the Prover's costs depend only on m. On the other hand, DT's Verifier has a larger cost than Protostar's, cf. Table 1. Since we are interested in folding schemes with a Prover sublinear on N and a succinct Verifier, we compare FLI mostly with Protostar and DT.

As we mentioned, FLI+SOS can naturally leverage the SOS decomposability of the big table $\mathbf{t}$, and as far as we are aware, FLI+SOS is the first of its kind in this sense. It is easy, however, to envision ways in which the previously mentioned schemes (Hypernova, Protostar, and DT) can also exploit SOS decomposability. Namely, one can first run the first step of Lasso (see Section 2), which splits a lookup instance into α smaller lookup instances, and then use either scheme to fold the α instances into α accumulated instances. We refer to this variation as $\{Scheme\}$ + SOS, where "$Scheme$" can be any folding scheme for lookup instances, e.g. Protostar, or DT. For Protostar, we consider that the logUp [Hab22] lookup argument is applied to the α instances, and for DT we consider that the

Table 1: Comparison of FLI with other folding schemes for lookup instances. The costs are organized in commitment, group exponentiation, and field multiplication costs. We also display the number of rounds of each scheme. This coincides with the number of challenges sent by the Verifier. m and N denote the size of the small table $\mathbf{a}$ on of the big table, respectively. In the "commit" column, a pair (n, S) refers to a commitment of a vector of size n with entries in the set S. FLI has two commitment cost profiles: one is average case, while the other is worst case (cf. Section 2 for further details). Here n_f we denote the number of foldings performed so far, u is a small parameter (denoted $c-1$ in [BC24]), and M denotes the maximum size of an entry in $\mathbf{t}$. The cost $\mathsf{P}_{\mathsf{sps}}$ refers to the field operation cost of running DT's underlying special sound protocol. In Appendix D this cost is approximated to $\alpha \cdot 19 \max\{m, N^{1/c}\}$ field multiplications. The cost L refers to the cost of computing the coeffients of the polynomial $e(X)$ in [BC24].
The efficiency of Protostar is displayed for their special-sound version of the logUp lookup argument. The efficiency of Hypernova is displayed for the lookup argument nlookup in [KS24]. When it comes to DT, we consider only the variant built upon logUp-GKR.

Scheme	**Prover work**			**Verifier work**		**Rounds**
	commit	**group**	**field**	**group**	**field**	
Protostar [BC23]	$(2m, \mathbb{F}), (m, [m])$	7	$O(m)$	3	$O(1)$	1
Hypernova [KS24]	–	–	$O(N)$	–	$O(m \log N)$	$\log(m) + O(1)$
Deep Thought [BC24]	$(3m, [M])$	$u \log N$	$\begin{cases} O(m \log(m)) + \\ + \mathsf{P}_{\mathsf{sps}} + \mathsf{L} \end{cases}$	$u \log N$	$u \log(N)$	$u \log(N)$
FLI (this work)	$\begin{cases} \text{avg: } (\rho, \mathbb{F}), (m-\rho, \mathbb{B}) \\ \text{worse: } (m, \mathbb{F}) \end{cases}$ $\rho := \min\{mn_f/N, m\}$	4	$O(m)$	4	$O(1)$	1

logUp-GKR [PH23] is applied to the α instances. We remark that proving accumulated instances for these schemes (Protostar + SOS and DT+SOS) is not a straightforward task. For the sake of comparison, we sketch a simplified method in Section 6. We compare our folding scheme FLI with Hypernova, Protostar, and DT; and we compare FLI+SOS with Protostar + SOS, and DT + SOS, cf. Tables 1 to 4 and Section 6.

Remark 1.1 (Using Protogalaxy)**.** Protogalaxy [EG23] continues the line of work of Protostar, and focuses on building a (multi) folding scheme for special-sound protocols with algebraic Verifiers in which the marginal folding Verifier work is really light. In particular, the authors manage to remove the commitment to the cross-term.

It is also easy to imagine a version of Protogalaxy+SOS in our terminology. However, in this particular case where we are using Protostar to fold logUp instances, Protostar is able to update the commitment to the error term homomorphically. This makes it such

Table 2: Comparison of the Prover in FLI+SOS with the Prover of other folding schemes when the big table $\mathbf{t}$ is SOS-decomposable (Definition 3.6). "Protostar + SOS" and "Deep Thought (DT) + SOS" both refer to first performing Lasso's SOS reduction and then applying Protostar or [BC24], respectively, to the resulting α lookup instances of a table of size m into a table of size $N^{1/c}$. We follow the same notation as in Table 1. $N^{1/c}$ is the size of the SOS decomposed tables (cf. Section 3.4); $\alpha = k \cdot c$, where k is a small constant (typically 1 or 2); and g is the multilinear polynomial providing the SOS decomposition and $|g|$ its arithmetic complexity.

Scheme	**Prover work**		
	commit	**group**	**field**
Protostar+SOS	$\alpha \cdot (2m, \mathbb{F}), \alpha \cdot (m, [m])$	7α	$m \deg(g)(\alpha + \lvert g\rvert)$
Deep Thought+SOS	$\alpha \cdot (3m, [M])$	$\alpha \cdot u \log(N^{1/c})$	$\begin{cases} O(\alpha m \log(m)) \\ +m \deg(g)(\alpha + \lvert g\rvert) \\ +\alpha \cdot \mathsf{P}_{\mathsf{sps}} + \alpha \cdot \mathsf{L} \end{cases}$
FLI+SOS	$\begin{cases} \text{avg: } c \cdot (\rho, \mathbb{F}), c \cdot (m - \rho, \mathbb{B}) \\ \text{worse: } c \cdot (m, \mathbb{F}) \end{cases}$ $c \cdot (m, \mathbb{B})$ $\rho := \min\{mn_f/N^{1/c}, m\}$	$4c + 1$	$m \deg(g)(\alpha + \lvert g\rvert)$

that the dominant cost for the folding Prover both in Protostar and Protogalaxy is the commitment to the witness. Since the constraint system is the same for Protostar and Protogalaxy, the cost to prove accumulated instances is also the same. This is why we only consider our variation of Protostar+SOS when comparing state-of-the-art folding schemes with FLI+SOS.

1.3 Organization of the paper

Section 2 outlines the techniques used to develop FLI. In Section 3, we introduce lookup relations and SOS-decomposable tables. Additional definitions on folding schemes, IPs/(P)IOPs, soundness, and the sumcheck protocol are in Appendix A. Section 4 constructs a PIOP and folding scheme for relation $\mathbf{R}_{elem}$ (stating that a matrix has elementary basis vectors as rows), which is extended in Section 5 to build FLI, incorporating a variation on SOS decomposition. Section 6 shows FLI's efficiency for key parameters. Deferred proofs are in

Table 3: Comparison of the Verifier in FLI+SOS with the Verifier of other folding schemes when the big table $\mathbf{t}$ is SOS-decomposable (Definition 3.6). We follow the same terminology and notation as in Tables 1 and 2

Scheme	**Verifier work**		**Rounds**
	group	**field**	
Protostar+SOS	3α	$O(\alpha \log(m))$	$\log(m) + \alpha$
Deep Thought+SOS	$\alpha u \log(N^{1/c})$	$O(\alpha \cdot u \log(N^{1/c}) + \log(m))$	$\log(m) + \alpha \cdot u \log(N^{1/c})$
FLI+SOS	$4c + 1$	$O(\alpha \log(m))$	$\log(m) + \alpha$

Appendix C, further comparisons in Appendix B, and Prover cost computations in Appendix D.

2 Techniques

Let $\mathbf{R}$ and $\mathbf{R}_{\mathsf{acc}}$ be two instance-witness relations. A folding scheme from $\mathbf{R} \times \mathbf{R}_{\mathsf{acc}}$ to $\mathbf{R}_{\mathsf{acc}}$ is an interactive protocol between a Prover and a Verifier. The Verifier takes as input a pair of instances $(\mathbb{x}, \mathbb{x}_{\mathsf{acc}}) \in \mathbf{R} \times \mathbf{R}_{\mathsf{acc}}$, and outputs a new instance $\mathbb{x}'_{\mathsf{acc}} \in \mathbf{R}_{\mathsf{acc}}$ at the end of the protocol. One requires that if the Prover knows a valid witness $\mathbb{w}'_{\mathsf{acc}}$ for $\mathbb{x}'_{\mathsf{acc}}$, then it knows (except with negligible probability) valid witnesses $\mathbb{w}$ and $\mathbb{w}_{\mathsf{acc}}$ for $\mathbb{x}$ and $\mathbb{x}_{\mathsf{acc}}$, respectively. Often, one speaks of folding schemes for $\mathbf{R}$, omitting $\mathbf{R}_{\mathsf{acc}}$.

Here we consider the *lookup relation* $\mathbf{R}_{\mathsf{Look}}$. For a fixed field $\mathbb{F}$, an instance-witness pair $(\mathbb{x}; \mathbb{w}) \in \mathbf{R}_{\mathsf{Look}}$ has the form $\mathbb{x} = (m, N, \mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}})$, and $\mathbb{w} = (\mathbf{a})$, where $m, N \in \mathbb{N}$, $\mathbf{t} \in \mathbb{F}^N, \mathbf{a} \in \mathbb{F}^m$, and $\mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}}$ are vector commitments to the vectors $\mathbf{t}, \mathbf{a}$. These vectors are often referred to as *tables*. The lookup instance is *valid* if $\{\mathbf{a}_i \mid i \in [m]\} \subseteq \{\mathbf{t}_i \mid i \in [N]\}$, and $\mathsf{Commit}(\mathbf{t}) = \mathsf{cm}_{\mathbf{t}}$, $\mathsf{Commit}(\mathbf{a}) = \mathsf{cm}_{\mathbf{a}}$ for a fixed commitment scheme Commit (for simplicity we omit referring to the randomness used in the commitments). In other words:

$$\mathbf{R}_{\mathsf{Look}} := \left\{ (\mathbb{x}; \mathbb{w}) = (m, N, \mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}}; \mathbf{a}) \;\middle|\; \begin{array}{l} \{\mathbf{a}_i \mid i \in [m]\} \subseteq \{\mathbf{t}_i \mid i \in [N]\} \\ \mathsf{Commit}(\mathbf{t}) = \mathsf{cm}_{\mathbf{t}}, \mathsf{Commit}(\mathbf{a}) = \mathsf{cm}_{\mathbf{a}} \end{array} \right\}$$

Informally, in this overview we sometimes denote elements in $\mathbf{R}_{\mathsf{Look}}$ by $(\mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}}; \mathbf{a}) \in \mathbf{R}_{\mathsf{Look}}$, omitting any reference to the table sizes m and N. Typically, we assume m and N to be powers of two, with N being much larger than m. Because of this, we often informally call $\mathbf{t}$ the *big table*, and $\mathbf{a}$ the *small table*.

Table 4: Dominant costs of the protocols for proving accumulated instances with FLI and FLI+SOS (cf. 5.4.2). We follow the same notation as in Tables 1 to 3. Besides that, $s \leq mN^{1/c}$ denotes the sparsity of the accumulated matrices $M_{\mathsf{acc}}, E_{\mathsf{acc}}, M_i^{\mathsf{acc}}$. The second and third column shows the dominant Prover and Verifier cost in field multiplications. In the Openings column, the notation $v \cdot (a\text{-variate}, b\text{-sparse})$ refers to v opening proofs of a-variate multilinear polynomials that are b-sparse (i.e. at most b of their evaluations in $\{0,1\}^{\log(a)}$ are nonzero). The Verifier opening proof costs are not reflected in the table. By "SOS" we mean that the multilinear polynomial is a small table resulting from a SOS decomposition. These can often be evaluated in $\log(N^{1/c})$ time, and hence the opening can be computed directly by the Verifier, rather than proved. By "dense" we mean that the polynomial can potentially take nonzero values on all the hypercube.

Scheme	**Prover field work**	**Verifier field work**	**Openings**
FLI	$(2m+1)N + s(\log(m)+3)$	$O(\log(mN))$	$1 \cdot (\log(mN)\text{-var}, \nu\text{-sparse})$ $1 \cdot (\log(N)\text{-var}, \text{SOS})$ $1 \cdot (\log(m)\text{-var}, \text{dense})$ $\nu := \min\{n_f m, mN\}$
FLI+SOS	$(2m+5\alpha+1)N^{1/c} +$ $+(3\alpha+2)s + 2m$	$O(\log(mN^{1/c}))$	$1 \cdot (\log(mN^{1/c})\text{-var}, \nu\text{-sparse})$ $1 \cdot (\log(N^{1/c})\text{-var}, \text{SOS})$ $1 \cdot (\log(m)\text{-var}, \text{dense})$ $\nu := \min\{n_f m, mN^{1/c}\}$

We next describe how FLI works at a high level. We first recall the now standard observation [ZBK+22, ZGK+22] that a lookup instance $(\mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}}; \mathbf{a}) \in \mathbf{R}_{\mathsf{Look}}$ is valid if and only if there exists a $m \times N$ matrix $M \in \mathbb{F}^{m \times N}$ such that:

- $M \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}^{\mathsf{T}}$, where T denotes transposition and $M \cdot \mathbf{t}^{\mathsf{T}}$ denotes matrix-vector multiplication.
- Each row of M is an *elementary basis vector*, i.e. it consists only of zeros, except for one entry, which is 1. We define a relation capturing this property:

$$\mathbf{R}_{\mathsf{elem}} := \left\{ (\mathbb{x}; \mathbb{w}) = (m, N, \mathsf{cm}_M; M) \;\middle|\; \begin{array}{l} \text{All rows of } M \text{ are elementary basis vectors,} \\ \mathsf{Commit}(M) = \mathsf{cm}_M \end{array} \right\}$$

- $\mathsf{Commit}(\mathbf{t}) = \mathsf{cm}_{\mathbf{t}}$ and $\mathsf{Commit}(\mathbf{a}) = \mathsf{cm}_{\mathbf{a}}$.

One can then define:

$$\mathbf{R}_{\mathsf{MLook}} := \left\{ \begin{pmatrix} \mathbb{x}; \\ \mathbb{w} \end{pmatrix} = \begin{pmatrix} m, N, \mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}}, \mathsf{cm}_M; \\ \mathbf{a}, M \end{pmatrix} \middle| \begin{array}{l} M \in \mathbb{F}^{m \times N} \\ (m, N, \mathsf{cm}_M; M) \in \mathbf{R}_{\mathsf{elem}} \\ M \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}^{\mathsf{T}} \\ \mathsf{Commit}(\mathbf{t}) = \mathsf{cm}_{\mathbf{t}}, \\ \mathsf{Commit}(\mathbf{a}) = \mathsf{cm}_{\mathbf{a}} \end{array} \right\}$$

As with $\mathbf{R}_{\mathsf{Look}}$, we will omit referring to m, N when talking about instance-witness pairs from $\mathbf{R}_{\mathsf{MLook}}$, and we proceed similarly with $\mathbf{R}_{\mathsf{elem}}$. Let $(\mathbb{x}_i; \mathbb{w}_i) = (\mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}_i}, \mathsf{cm}_{M_i}; \mathbf{a}_i, M_i)$ $(i = 1, 2$)be two instance-witness pairs from $\mathbf{R}_{\mathsf{MLook}}$ that we wish to fold. By definition, we have $M_1 \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}_1{}^{\mathsf{T}}$ and $M_2 \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}_2{}^{\mathsf{T}}$. We visualize such instances as:

$$\begin{cases} M_1 \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}_1^{\mathsf{T}} \\ (\mathsf{cm}_{M_1}; M_1) \in \mathbf{R}_{\mathsf{elem}} \end{cases} \qquad \begin{cases} M_2 \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}_2^{\mathsf{T}} \\ (\mathsf{cm}_{M_2}; M_2) \in \mathbf{R}_{\mathsf{elem}} \end{cases} \tag{1}$$

Note that, fixing $\mathbf{t}$, the first part of the instances in (1) are linear constraints on the matrices $M_1, \mathbf{a}_1$ and $M_2, \mathbf{a}_2$. Hence, a natural step towards folding the instances (1) is to have the verifier send a random challenge $\alpha \leftarrow \mathbb{F}$, and then merge (1) into a single claim of the form

$$\begin{cases} (M_1 + \alpha M_2) \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}_1^{\mathsf{T}} + \alpha \mathbf{a}_2^{\mathsf{T}} \\ (\mathsf{cm}_{M_1}; M_1), (\mathsf{cm}_{M_2}; M_2) \in \mathbf{R}_{\mathsf{elem}} \end{cases} \tag{2}$$

The next natural step is to apply a folding scheme for the relation $\mathbf{R}_{\mathsf{elem}}$, so that the two claims above, namely $(\mathsf{cm}_{M_1}; M_1) \in \mathbf{R}_{\mathsf{elem}}$ and $(\mathsf{cm}_{M_2}; M_2) \in \mathbf{R}_{\mathsf{elem}}$, can be folded into an instance of an accumulated version of the relation $\mathbf{R}_{\mathsf{elem}}$. One way to do this is by first noting that $(\mathsf{cm}_M; M)$ belongs to $\mathbf{R}_{\mathsf{elem}}$ if and only if:

- $M_{ij}^2 = M_{ij}$ for all matrix entries M_{ij} of M $(i \in [m], j \in [N])$. This property is equivalent to saying that all entries of M are either 0 or 1.
- $M \cdot \mathbf{1}^{\mathsf{T}} = \mathbf{1}^{\mathsf{T}}$, where $\mathbf{1}$ is the N-dimensional vector consisting entirely of 1's. Together with the above property, this ensures that all rows of M are elementary vectors[1].
- $\mathsf{Commit}(M) = \mathsf{cm}_M$.

By looking at M as a mN-dimensional witness vector, we reformulate the above conditions as a R1CS-type constraint. Then, we use a Nova-like approach [KST22] so as to obtain a folding scheme for the relation $\mathbf{R}_{\mathsf{elem}}$ into a relaxed version of $\mathbf{R}_{\mathsf{elem}}$, which we denote $\mathbf{R}_{\mathsf{elem}}^{\mathsf{acc}}$. Namely $\mathbf{R}_{\mathsf{elem}}^{\mathsf{acc}}$ incorporates a mN-dimensional error vector E, a slackness parameter μ, and

[1] Here one needs to assume that the characteristic of $\mathbb{F}$ is larger than N

a commitment cm_E to E. Then $(\mu, \mathsf{cm}_M, \mathsf{cm}_E; M, E) \in \mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}}$ if and only if $M_{ij}^2 = M_{ij} + E_{ij}$ for all $i \in [m], j \in [N]$, $M \cdot \mathbf{1}^T = (1+\mu) \cdot \mathbf{1}^T$, and the commitments cm_M and cm_E are commitments to M and E. Formally,

$$\mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}} := \left\{ \begin{pmatrix} \mathbb{x}; \\ \mathbb{w} \end{pmatrix} = \begin{pmatrix} m, N, \mu, \mathsf{cm}_M, \mathsf{cm}_E; \\ M, E \end{pmatrix} \;\middle|\; \begin{array}{l} M \circ M = M + E \\ M \cdot \mathbf{1}^{\mathsf{T}} = (1+\mu) \cdot \mathbf{1}^{\mathsf{T}}, \\ \mathsf{Commit}(M) = \mathsf{cm}_M, \mathsf{Commit}(E) = \mathsf{cm}_E \end{array} \right\}$$

Here $\circ$ denotes the Hadamard (i.e. component-wise) product. Given $(\mathsf{cm}_M; M) \in \mathbf{R}_{\mathsf{elem}}$ and $(\mu, \mathsf{cm}_{M_{\mathsf{acc}}}, \mathsf{cm}_E; M_{\mathsf{acc}}, E) \in \mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}}$:

1. The Prover computes a commitment cm_T to an intermediate cross term $T = 2(M_{\mathsf{acc}} \circ M) - M$ and sends cm_T to the Verifier;
2. The Verifier replies with a random challenge $\alpha \leftarrow \mathbb{F}$;
3. Both Prover and Verifier output $\mathsf{cm}_{M'_{\mathsf{acc}}} = \mathsf{cm}_{M_{\mathsf{acc}}} + \alpha \mathsf{cm}_M$, $\mathsf{cm}_{E'} = \mathsf{cm}_E + \alpha \mathsf{cm}_T + \alpha^2 \mathsf{cm}_M$, $\mu' = \mu + \alpha$;
4. The Prover additionally outputs $M'_{\mathsf{acc}} = M_{\mathsf{acc}} + \alpha M$, $E' = E + \alpha T + \alpha^2 M$.

As a result, FLI's folding Verifier is very simple, only performing a handful of operations with a given vector commitment. When using, say, curve-based commitment schemes, this translates to 4 group additions and 4 scalar multiplications. Note that FLI only needs one random oracle query. The folding proof size consist in one group and field elements (we don't count $\mathsf{cm}_{M'_{\mathsf{acc}}}, \mathsf{cm}_{E'}$ as part of the proof).

When it comes to Prover costs, note that the matrix $M \in \mathbb{F}^{m \times N}$ (which we look at as a vector of size mN) is *m-sparse*, meaning that all its entries except m are 0. In fact, all nonzero entries of M are 1. Consequently, the matrix $M_{\mathsf{acc}} \circ M$ is also m-sparse, and so is the vector T. As such, standard curve-based commitment schemes allow to commit to M and to T in time $O(m)$. For example, the PST[2] [PST13, CGG+23] scheme can commit to M with exactly $m - 1$ group additions.

On the other hand, in general, nonzero elements in T can have arbitrary size. This is because M_{acc} has entries that are computed from the Verifier's previous folding challenges, which were sampled randomly in $\mathbb{F}$. The concrete cost of committing to T in the *worst case* can be relatively large, e.g. $\approx 2^8 m$ or $2^9 m$ when using PST and Pippenger's algorithm to compute Multi Scalar Multiplications (MSM) (see, for example, the benchmarks in Table 1 of [Hab22]). However, we remark that the above is a *worst case*. Since $M \in \mathbf{R}_{\mathsf{elem}}$, if M_{acc} is very sparse (as is when not many folding steps have been performed), then $M_{\mathsf{acc}} \circ M$

[2]Here we refer to the PST version from [CGG+23] which uses the Lagrange basis instead of the monomial basis.

is even more sparse with high likelihood. If we think of M as being randomly sampled in $\mathbf{R}_{\mathsf{elem}}$, then the number of nonzero elements in $M_{\mathsf{acc}} \circ M$ is, in expectation, $\leq mn_f/N$, where n_f is the number of folding steps performed so far. Thus, when $n_f m << N$, we have that $M_{\mathsf{acc}} \circ M$ is essentially the zero vector, on average. In that case, **FLI's Prover only commits to sparse vectors containing almost exclusively small entries**.

Lasso's SOS decomposability. We recall SOS-decomposability, one of the key ideas of the Lasso and Jolt papers [STW24, AST24]. Formally, a table $\mathbf{t} \in \mathbb{F}^N$ is *SOS-decomposable* if there exists $\alpha := k \cdot c$ tables $\mathbf{t}_1, \ldots, \mathbf{t}_\alpha$ of size $N^{1/c}$, i.e. $\mathbf{t}_i \in \mathbb{F}^{N^{1/c}}$ for all i, and an α-variate multilinear polynomial g such that:

$$\forall \mathbf{y} \in \mathbb{B}^{\log N},\ t(\mathbf{y}) = g\left(\begin{array}{r}\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_k(\mathbf{y}_1), \mathbf{t}_{k+1}(\mathbf{y}_2), \ldots, \mathbf{t}_{2k}(\mathbf{y}_2), \ldots \\ \ldots, \mathbf{t}_{kc-1}(\mathbf{y}_c), \mathbf{t}_\alpha(\mathbf{y}_c)\end{array}\right) \tag{3}$$

Here, we let $\mathbb{B} = \{0, 1\}$, and we index the entries of $\mathbf{t}$ with elements from the hypercube $\mathbb{B}^{\log N}$, denoting $\mathbf{t}(\mathbf{y})$ the entry of $\mathbf{t}$ indexed by the element $\mathbf{y} \in \mathbb{B}^{\log(N)}$. Further, $\mathbf{y}_1, \ldots, \mathbf{y}_c$ are all vectors from $\mathbb{B}^{\log(N)/c}$ such that $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_c)$. For the applications mentioned in Lasso and Jolt [STW24, AST24], α is c or a small multiple of c. As exemplified by the Jolt paper [AST24], many natural tables $\mathbf{t}$ are SOS-decomposable (e.g., tables containing RISC-V instructions). Importantly for this paper, c can be chosen as large as wanted.

FLI and its natural use of SOS decomposability (FLI+SOS). One of the contributions of this work is a variation of SOS decompositions that blends seamlessly with our folding approach. We emphasize that this does not simply consist in performing Lasso's SOS decomposition and then folding the resulting smaller lookup instances. The starting observation is that when $\mathbf{t}$ is SOS-decomposable, the statement that $(\mathbf{t}, \mathsf{cm}_\mathbf{t}, \mathsf{cm}_\mathbf{a}; \mathbf{a}) \in \mathbf{R}_{\mathsf{Look}}$ is equivalent (leaving aside the constraints involving $\mathsf{cm}_\mathbf{t}$ and $\mathsf{cm}_\mathbf{a}$) to the existence of $m \times N^{1/c}$ matrices $M_1, \ldots, M_c$ with elementary vectors as rows such that:

$$\forall \mathbf{x} \in \mathbb{B}^{\log(m)},\ \mathbf{a}(\mathbf{x}) = g\left(\sum_{\mathbf{y}} M_1(\mathbf{x}, \mathbf{y}) \cdot \mathbf{t}_1(\mathbf{y}), \ldots, \sum_{\mathbf{y}} M_c(\mathbf{x}, \mathbf{y}) \cdot \mathbf{t}_\alpha(\mathbf{y})\right) \tag{4}$$

where $\mathbf{y}$ runs over $\mathbb{B}^{\log(N^{1/c})}$. Indeed, one has that $(\mathbf{t}, \mathsf{cm}_\mathbf{t}, \mathsf{cm}_\mathbf{a}; \mathbf{a}) \in \mathbf{R}_{\mathsf{Look}}$ if and only if for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$, there exists a $\mathbf{y} \in \mathbb{B}^{\log(N)}$ such that $\mathbf{a}(\mathbf{x}) = \mathbf{t}(\mathbf{y})$. But Eq. (3) implies that $\mathbf{t}(\mathbf{y}) = g(\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_k(\mathbf{y}_1), \mathbf{t}_{k+1}(\mathbf{y}_2), \ldots, \mathbf{t}_\alpha(\mathbf{y}_c))$, where $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_c) \in \mathbb{B}^{\log(N)/c}$. The matrices $M_1, \ldots, M_c$ respectively indicate, for each $\mathbf{x} \in \mathbb{B}^{\log(m)}$, the indices $\mathbf{y}_1, \ldots, \mathbf{y}_c$ such that $\mathbf{a}(\mathbf{x}) = g(\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_k(\mathbf{y}_1), \mathbf{t}_{k+1}(\mathbf{y}_2), \ldots, \mathbf{t}_\alpha(\mathbf{y}_c))$ holds, i.e. for each $\mathbf{x} \in \mathbb{B}^{\log(m)}$ and $i \in [c]$, the row of M_i indexed by the element $\mathbf{x}$ consists of zeros everywhere except for a one in the column indexed by $\mathbf{y}_i$. This way, $\sum_\mathbf{y} M_i(\mathbf{x}, \mathbf{y}) \cdot \mathbf{t}_j(\mathbf{y}) = \mathbf{t}_j(\mathbf{y}_i)$, for all $j \in [k]$.

Above and below, we look at $\mathbf{a}, M_i, \mathbf{t}_i$ as the multilinear extensions (MLE) (cf. Section 3.1) of the corresponding vectors.

With (4) in mind, we describe an extension of FLI that can handle SOS decomposability. Say we wish to fold two instances $(\mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}_i}; \mathbf{a}_i) \in \mathbf{R}_{\mathsf{Look}}$, $i = 1, 2$. The protocol, which we call FLI+SOS, proceeds as follows:

- P starts off by committing to the m-sparse matrices $M_{1,1}, \ldots, M_{1,c}$ and $M_{2,1}, \ldots, M_{2,c}$ such that (4) holds, respectively, for the instances $(\mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}_1}; \mathbf{a}_1)$ and $(\mathbf{t}, \mathsf{cm}_{\mathbf{t}}, \mathsf{cm}_{\mathbf{a}_2}; \mathbf{a}_2)$. Let $\mathsf{cm}_{M_{i,j}}$ be the commitments. Then $(\mathsf{cm}_{M_{i,j}}; M_{i,j}) \in \mathbf{R}_{\mathsf{elem}}$.
- Next for both $i = 1, 2$, P and V run a sumcheck protocol to assert the equality:

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} \left(\mathbf{a}_i(\mathbf{x}) - g\left(\sum_{\mathbf{y}} M_{i,1}(\mathbf{x}, \mathbf{y}) \cdot \mathbf{t}_1(\mathbf{y}), \ldots, \sum_{\mathbf{y}} M_{i,c}(\mathbf{x}, \mathbf{y}) \cdot \mathbf{t}_\alpha(\mathbf{y}) \right) \right) \widetilde{\mathsf{eq}}(\boldsymbol{\beta}, \mathbf{x}) = 0$$

  where $\boldsymbol{\beta} \in \mathbb{F}^{\log(m)}$ is a random challenge from the Verifier. This equality ensures that, except with negligible probability, (4) holds for $i = 1, 2$.
- At the end of the sumcheck protocol, P and V are left with evaluation claims of the following form, for $j \in [c], i \in \{1, 2\}$, and $(j-1)k + 1 \le \ell \le jk$:

$$\mathbf{a}_i(\mathbf{r}) = d, \quad \sum_{\mathbf{y}} M_{i,j}(\mathbf{r}, \mathbf{y}) \cdot \mathbf{t}_\ell(\mathbf{y}) = c_{ij\ell},$$

- The Verifier sends a random challenge $\alpha \in \mathbb{F}$, and the resulting folded instance is

$$(\mathbf{a}_1 + \alpha \mathbf{a}_2)(\mathbf{r}) = d', \quad (\mathsf{cm}_{M_{i,j}}; M_{i,j}) \in \mathbf{R}_{\mathsf{elem}}$$
$$\sum_{\mathbf{y}} (M_{1,j} + \alpha M_{2,j})(\mathbf{r}, \mathbf{y}) \cdot \mathbf{t}_\ell(\mathbf{y}) = c'_{j\ell}, \quad j \in [c], (j-1)k + 1 \le \ell \le jk$$

  for random $\mathbf{r} \in \mathbb{F}^{\log(m)}$ and some field elements $d', c'_{j\ell}$.
- As we explained, the claims $(\mathsf{cm}_{M_{i,j}}; M_{i,j}) \in \mathbf{R}_{\mathsf{elem}}$ can be expressed as R1CS instances, and folded in a Nova-like fashion.

This is a simplified description of FLI+SOS where both instances are lookup instances. In practice, we fold lookup instances into a relaxed lookup relation that we describe in Section 5. Further, we use a single matrix to accumulate all the claims of the form $(\mathsf{cm}_{M_{i,j}}; M_{i,j}) \in \mathbf{R}_{\mathsf{elem}}$, and in parallel we fold the evaluation claims. The folding Prover and Verifier of FLI+SOS are still the cheapest (see Tables 2 and 3). We also show by using an example that the cost of proving accumulated instances with FLI is the cheapest in relevant scenarios (Section 6).

Proving accumulated instances. We describe custom PIOPs that allow to prove that an instance accumulated with FLI+SOS (or FLI) is valid. These protocols are simple and only consist in sumchecks, cf. Table 4, and Section 6. The dominant Prover cost for proving accumulated instances with FLI+SOS is $m(2N^{1/c}+1)+3\alpha\cdot m\cdot n_f$ field operations, with the $mN^{1/c}$ cost being due to the fact that we treat the $m\times N^{1/c}$ matrices M_i as dense matrices, and the other cost is related to the sparsity of the accumulated matrices after n_f foldings. The fact that we can freely choose the parameter c makes this Prover cost very similar to the other available options in some practical scenarios, which we discuss in Section 6). This is considering very optimistic costs for the protocols that prove accumulated instances with Protostar+SOS or DT+SOS (see Section 6, and Remark 6.2). The final step in the protocols that prove accumulated instances is the opening of certain multivariate polynomials in $\log(m)$ or $\log(mN^{1/c})$ variables at random vectors of field elements. While these openings can be expensive, many of them are at the same vector of field elements and can be batched. By choosing a polynomial commitment scheme carefully (like certain variations [Tha24] of Zeromorph [KT23]), it is possible to reduce the cost of these openings at the expense of a worse Verifier cost. We discuss this in Section 6.

3 Preliminaries

Throughout the document we fix a finite field $\mathbb{F}$. Given an integer $k\geq 1$ we let $[k]:=\{1,\ldots,n\}$. We let $\mathbb{B}^k=\{0,1\}^k:=\{(b_1,\ldots,b_k)\mid b_i\in\mathbb{B},\text{ for all } i\in[k]\}$ be the hypercube of dimension k, or, in other words, the set of all sequences of k bits. We next provide formal descriptions pertaining multilinear polynomials, lookup relations, folding schemes, and SOS decomposability. We refer to Appendix A for extended preliminaries on interactive proofs, and the sumcheck protocol.

Let $\mathbf{X}=(X_1,\ldots,X_n)$ be a vector of variables. We let $\mathbb{F}[\mathbf{X}]$ denote the ring of multivariate polynomials on variables $\mathbf{X}$ and with coefficients in $\mathbb{F}$. By $\mathbb{F}^{\leq d}[\mathbf{X}]$ we denote the set of polynomials from $\mathbb{F}[\mathbf{X}]$ whose variables have individual degree at most d. For example, $\mathbb{F}^{\leq 1}[\mathbf{X}]$ is the set of multilinear polynomials on variables $\mathbf{X}$.

We use λ to denote the security parameter. A function $f(\lambda)$ is in $\mathsf{poly}(\lambda)$ if there exists a $c\in\mathbb{N}$ such that $f(\lambda)=O(\lambda^c)$. If for all $c\in\mathbb{N}$, $f(\lambda)=o(\lambda^{-c})$, then $f(\lambda)$ is in $\mathsf{negl}(\lambda)$ and is said to be negligible.

3.1 Multilinear Polynomials

Let $n\geq 1$ and let $\mathbf{X}=(X_1,\ldots,X_n)$ be a tuple of variables. It is well-known that a multilinear polynomial $f(\mathbf{X})\in\mathbb{F}^{\leq 1}[\mathbf{X}]$ is uniquely defined by the multiset of the values it takes on $\mathbb{B}^n$, i.e. $f(\mathbb{B}^n):=\{f(\mathbf{x})\mid\mathbf{x}\in\mathbb{B}^n\}$. In other words, any two $f,g\in\mathbb{F}^{\leq 1}[\mathbf{X}]$ such that $f(\mathbf{x})=g(\mathbf{x})$ for all $\mathbf{x}\in\mathbb{B}^n$ are the same polynomial. Further, given a map $f:\mathbb{B}^n\to\mathbb{F}$, there always exist a unique multilinear polynomial on n variables, denoted $\widetilde{f}(\mathbf{X})$, such that

$\widetilde{f}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{B}^n$. It is given by the expression

$$\widetilde{f}(\mathbf{X}) := \sum_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \mathbf{X}) \tag{5}$$

where $\widetilde{\mathsf{eq}}(\mathbf{x}; X)$ is the unique multilinear polynomial on n variables that takes the value 0 on all points of the hypercube $\mathbb{B}^n$, except at $\mathbf{x}$ where it takes the value 1. Precisely,

$$\widetilde{\mathsf{eq}}(\mathbf{x}; \mathbf{X}) := \prod_{i \in [n]} \left(x_i X_i - (1 - x_i)(1 - X_i)\right).$$

This unique multilinear polynomial $\widetilde{f}(\mathbf{X})$ is called the *multilinear extension (MLE)* of f. Given a vector $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{F}^N$, we define the MLE of $\mathbf{v}$ (denoted by $\widetilde{\mathbf{v}}(\mathbf{X})$) as the MLE of the map $\mathbf{v} : \mathbb{B}^n \to \mathbb{F}$ assigning to each element $\mathbf{x} \in \mathbb{B}^n$ the element $v_{\mathbf{x}}$, where here we interpret $\mathbf{x}$ as the natural number whose binary representation is $\mathbf{x}$.

3.2 Multilinear Polynomial Commitment Schemes

In this paper we use multilinear polynomial commitments schemes (PCSs), a class of commitment schemes that work with multilinear polynomials. As we explained, one can see vectors in $\mathbb{F}^N$ as multilinear polynomials $\mathbb{B}^{\log(N)} \to \mathbb{F}$, and in this way multilinear PCSs allows a Prover to commit to vectors as well.

Definition 3.1 (Multilinear PCS)**.** *A multilinear polynomial commitment scheme* PC *over a field* $\mathbb{F}$ *consists of a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$*:*

- $\mathsf{Setup}(1^\lambda, s) \to \mathrm{pp}$ *takes security parameter* λ *and* $s \in \mathbb{N}$ *(i.e. the number of variables in the polynomials), and outputs public parameters* pp*.*
- $\mathsf{Commit}(\mathrm{pp}, f) \to C$ *takes a multilinear polynomial* $f \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_s]$ *and outputs a commitment* C*.*
- $\mathsf{Open}(\mathrm{pp}, C, f) \to b$ *takes a commitment* C *and a multilinear polynomial* $f \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_s]$*, and outputs a bit* b*.*
- $\mathsf{Eval}(\mathrm{pp}, C, \mathbf{z}, y; f)\langle \mathsf{P}, \mathsf{V}\rangle \to b$ *is an interactive public-coin protocol between a PPT Prover* P *and Verifier* V *with public input a commitment* C*, an evaluation point* $\mathbf{z} \in \mathbb{F}^s$ *and a value* $y \in \mathbb{F}$*.* P *additionally knows a multilinear polynomial* $f \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_s]$ *and* P *wants to convince* V *that* f *is an opening of* C *and* $f(\mathbf{z}) = y$*. The Verifier outputs a bit* b *at the end of the protocol.*

A multilinear PCS PC *is said to be* succinct *if the output of* Commit *is sublinear in* 2^s*. It is said to be* binding *if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\left[b_0 = b_1 = 1 \wedge f_0 \neq f_1 \;\middle|\; \begin{array}{r} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, s) \\ (C, f_0, f_1) \leftarrow \mathcal{A}(\mathrm{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathrm{pp}, C, f_0) \\ b_1 \leftarrow \mathsf{Open}(\mathrm{pp}, C, f_1) \end{array}\right] \leq \mathrm{negl}(\lambda)$$

If Eval is an argument of knowledge, then the polynomial commitment scheme is said to be knowledge-sound, or *extractable*. We define this notion formally.

Definition 3.2 (Extractable multilinear PCS)**.** *A multilinear PCS is said to be extractable if* Eval *is an argument of knowledge in the following sense. Consider the relation:*

$$\mathbf{R}_{\mathsf{Eval},\mathrm{pp}} = \{(\mathbb{x}; \mathbb{w}) = ((C, \mathbf{z}, y); f) \mid f(\mathbf{z}) = y \wedge \mathsf{Open}(\mathrm{pp}, C, f) = 1\},$$

then, there is an extractor Ext *such that for any PPT algorithm* $\mathcal{A}$ *where*

$$\Pr[\mathsf{Eval}(\mathrm{pp}, C, \mathbf{z}, y; f)\langle \mathcal{A}, \mathsf{V}\rangle = 1 \mid (C, \mathbf{z}, y, \pi) \leftarrow \mathcal{A}(\mathrm{pp})] \geq \varepsilon(\lambda)$$

for some non-negligible function ε*,* $\mathsf{Ext}^{\mathcal{A}}(\mathbb{x})$ *with black-box access to* $\mathcal{A}$ *can output a witness* $f \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_s]$ *in expected polynomial time such that* $((C, \mathbf{z}, y); f) \in \mathbf{R}_{\mathsf{Eval},\mathrm{pp}}$ *with overwhelming probability.*

Lastly, we define the notion of additively homomorphic PCS. This is useful in the context of folding as it allows one to update the commitment of a linear combination of polynomials without needing to recompute the commitment.

Definition 3.3 (Additively homomorphic PCS)**.** *Let* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *be a multilinear PCS, and suppose that the target set of the* Commit *algorithm is an additive group* $\mathbb{G}$*.* PC *is said to be additively homomorphic if for any* $f, g \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_s]$*, it holds that:*

$$\mathsf{Commit}(\mathrm{pp}, f + g) = \mathsf{Commit}(\mathrm{pp}, f) + \mathsf{Commit}(\mathrm{pp}, g)$$

Throughout the paper we fix a Polynomial Commitment Scheme (PCS) for multilinear polynomials $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ (cf. Section 3.2).

3.3 Lookup relations

An *indexed relation* is a subset $\mathbf{R} \subseteq \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$. Given $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}$, the string $\mathbb{i}$ is called an *index*, $\mathbb{x}$ is called an *instance*, and $\mathbb{w}$ a *witness*. In this paper we often interpret instances and witnesses as vectors of field elements and indices consisting of tuples of natural numbers and field descriptions, but this need not always be the case. When

describing Polynomial IOPs for example (see Appendix A.1), the index and the instance can contain oracles to polynomials, and the witness can contain polynomials.

Throughout the paper we let N denote a "large table" size and $m \leq N$ denote a "small table" size. For simplicity, we assume both N and m are powers of 2. A vector $\mathbf{v} \in \mathbb{F}^k$ is said to be *r-sparse* if $\mathbf{v}$ has at most r entries different than 0.

Definition 3.4 (Lookup relation and big/small tables)**.** *The* lookup relation $\mathbf{R}_{\mathsf{Look}}$ *is defined as:*

$$\mathbf{R}_{\mathsf{Look}} := \left\{ \begin{pmatrix} \mathbb{i} = (\mathbb{F}, N, m), \\ \mathbb{x} = (\mathbf{a}, \mathbf{t}); \\ \mathbb{w} = \emptyset \end{pmatrix} \;\middle|\; \begin{array}{l} N, m \geq 1, \\ \{a(\mathbf{x}) \mid \mathbf{x} \in \mathbb{B}^{\log m}\} \subseteq \{t(\mathbf{y}) \mid \mathbf{y} \in \mathbb{B}^{\log(N)}\}. \end{array} \right\}$$

We call $\mathbf{a}$ *a* small table*, and* $\mathbf{t}$ *a* lookup table *(or big table).*

Unless stated otherwise, we make no assumption on the number of repeated values in $\mathbf{t}$. I.e. $\mathbf{t}$ may have repeated values.

Definition 3.5 (Lookup instances)**.** *We call a tuple of the form* $((\mathbb{F}, N, m), (\mathbf{a}, \mathbf{t}))$ *with* $\mathbf{t} \in \mathbb{F}^N$*,* $\mathbf{a} \in \mathbb{F}^m$ *a* lookup instance*. Sometimes the index* $(\mathbb{F}, N, m)$ *is omitted. An instance may or may not belong to* $\mathbf{R}_{\mathsf{Look}}$*.*

Committed matrix lookup relations. A now standard observation [ZBK+22, ZGK+22] is that a tuple $((\mathbb{F}, N, m), (\mathbf{a}, \mathbf{t}))$ is in $\mathbf{R}_{\mathsf{Look}}$ if and only if there exists a matrix $M \in \mathbb{F}^{m \times N}$ such that:

- $M \cdot \mathbf{t}^\mathsf{T} = \mathbf{a}^\mathsf{T}$,
- The rows of M are vectors in the standard basis of $\mathbb{F}^N$. This second condition is equivalent, when $N < \mathrm{char}(\mathbb{F})$, to the equations $M \circ M = M$ ($\circ$ denotes the Hadamard product) and $M \cdot \mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T}$.

It is convenient to translate these last two conditions and express them as relationships between multilinear polynomials, by using the MLEs of the vectors $\mathbf{a}, \mathbf{t}$ and M. A tuple $((\mathbb{F}, N, m), (\mathbf{a}, \mathbf{t}))$ is in $\mathbf{R}_{\mathsf{Look}}$ if and only if there exists a $\log(m) + \log(N)$-variate multilinear polynomial M such that:

- $\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}(\mathbf{X})$
- For all $\mathbf{x}, \mathbf{y}$, $M(\mathbf{x}, \mathbf{y})^2 = M(\mathbf{x}, \mathbf{y})$; and $\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{X}, \mathbf{y}) = 1$. By abuse of notation, we still write the first condition as $M \circ M = M$, and the second one as $M \cdot \mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T}$.

Further, the situation is often such that the Prover commits to the tables $\mathbf{a}, \mathbf{t}$[3], and has additional witnesses for these commitments. We let $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ be a multilinear PCS. For simplicity, we abbreviate Commit as cm. We define the committed matrix algebraic[4] lookup relation as follows:

$$\mathbf{R}_{\mathsf{CmMAILook}} := \left\{ \begin{pmatrix} (\mathbb{F}, N, m, \mathbf{t}), \\ (\overline{\mathbf{a}}, \overline{M}); \\ (\mathbf{a}, M) \end{pmatrix} \;\middle|\; \begin{array}{c} N < \mathrm{char}(\mathbb{F}), \\ M \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_{\log(m)}, Y_1, \ldots, Y_{\log(N)}], \\ \mathbf{a} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_{\log(m)}], \\ \mathbf{t} \in \mathbb{F}^{\leq 1}[Y_1, \ldots, Y_{\log(N)}], \\ \overline{M} = \mathsf{cm}(M),\ \overline{\mathbf{a}} = \mathsf{cm}(\mathbf{a}), \\ \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}(\mathbf{X}), \\ M \circ M = M,\ M \cdot \mathbf{1}^{\mathsf{T}} = \mathbf{1}^{\mathsf{T}} \end{array} \right\}$$

This rewriting of the lookup relation will be useful for us when formally describing our folding scheme in Section 5.

3.4 SOS-decomposable tables

We recall the definition of SOS decomposition from [STW24].

Definition 3.6 (SOS decomposition)**.** *Let $c, k \geq 1$, $\alpha = k \cdot c$, and let $\mathbf{t} \in \mathbb{F}^N$ be a table of size N. Assume $N^{1/c}$ is a power of two. Let $\mathbf{t}_1, \ldots, \mathbf{t}_\alpha \in \mathbb{F}^{N^{1/c}}$ be α tables of size $N^{1/c}$. We say that $\mathbf{t}$ admits a* SOS decomposition *with respect to the tables $\mathbf{t}_1, \ldots, \mathbf{t}_\alpha$ if there exists a multilinear polynomial $g = g(Y_1, \ldots, Y_\alpha) \in \mathbb{F}[Y_1, \ldots, Y_\alpha]$ in α variables such that:*

$$\forall \mathbf{y} \in \mathbb{B}^{\log(N)},\ \mathbf{t}(\mathbf{y}) = g(\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_k(\mathbf{y}_1), \mathbf{t}_{k+1}(\mathbf{y}_2), \ldots, \mathbf{t}_\alpha(\mathbf{y}_c))$$

where $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_c) \in (\mathbb{B}^{\log(N)/c})^c$, and further each $\mathbf{t}_i$ can be evaluated in $O(\log(N)/c)$ field operations at any $\mathsf{r} \in \mathbb{F}^{\log(N)/c}$.

3.5 Folding schemes

We recall the notion of a folding scheme in the sense of [EG23]. We use the convention that if we say that the protocol has input $(a; b)$, both the Prover and Verifier get a, but only the Prover gets b. The definition is slightly adapted to our case, where the relation and the accumulation relation share the index.

[3] Unless $\mathbf{t}$ has a particular type of structure, e.g., the SOS structure.

[4] The term "algebraic" refers to the fact that the conditions relating $M, \mathbf{a}, \mathbf{t}$ are expressed with multilinear polynomials.

Definition 3.7. *Fix indexed relations* $\mathbf{R}$ *and* $\mathbf{R}_{\mathsf{acc}}$*. An* $(\mathbf{R} \to \mathbf{R}_{\mathsf{acc}})$*-folding scheme is a public-coin interactive protocol* $\mathscr{P}$ *between a Prover* P *and a Verifier* V *such that:*

1. *The protocol input is* $(\mathbb{i}, \mathbb{x}, \mathbb{x}'; \mathbb{w}, \mathbb{w}')$
2. *When the protocol ends,* V *outputs* $\mathbb{x}^*$*, and* P *outputs* $\mathbb{w}^*$
3. ***(Perfect) Completeness:*** *If* $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}_{\mathsf{acc}}$*,* $(\mathbb{i}, \mathbb{x}'; \mathbb{w}') \in \mathbf{R}$*, and* P, V *are honest, then* $(\mathbb{i}, \mathbb{x}^*; \mathbb{w}^*) \in \mathbf{R}_{\mathsf{acc}}$ *with probability one.*
4. ***Knowledge soundness:*** *The following protocol* $\widetilde{\mathscr{P}}$ *between* $\widetilde{\mathsf{P}}, \widetilde{\mathsf{V}}$ *for the relation* $\mathbf{R}_{\mathsf{acc}} \times \mathbf{R}$ *is knowledge sound with error negligible in the security parameter:*
   (a) *Given inputs* $(\mathbb{i}, \mathbb{x}, \mathbb{x}'; \mathbb{w}, \mathbb{w}')$*,* $\widetilde{\mathsf{P}}, \widetilde{\mathsf{V}}$ *run* $\mathscr{P}$ *as* P, V *on the same inputs.*
   (b) *Let* $(\mathbb{i}, \mathbb{x}^*; \mathbb{w}^*)$ *be the final output of* P, V *in* $\mathscr{P}$*.* $\widetilde{\mathsf{V}}$ *accepts if and only if* $(\mathbb{i}, \mathbb{x}^*; \mathbb{w}^*) \in \mathbf{R}_{\mathsf{acc}}$*.*

We call instances for the relation $\mathbf{R}_{\mathsf{acc}}$ folded *or* accumulated instances*.*

This definition allows us to enlarge the relation $\mathbf{R}_{\mathsf{CmMAILook}}$ to a slightly more general one, so that we can apply the folding step. We state the following slight generalization of a lemma in [KST22] in the sense that it need not be the case that $\mathbf{R}_{\mathsf{acc}}$ and $\mathbf{R}$ are identical, but it follows by the same arguments as in [KST22].

Lemma 3.1 (Forking Lemma for Folding Schemes, Lemma 1 in [KST22])**.** *Consider a* $(2\mu + 1)$*-move* $(\mathbf{R} \to \mathbf{R}_{\mathsf{acc}})$*-folding scheme* Π*. The protocol* Π *satisfies knowledge soundness if there exists a PPT algorithm* Ext *such that for all input tuples* $(\mathbb{i}, \mathbb{x}, \mathbb{x}')$*, outputs witnesses* $(\mathbb{w}, \mathbb{w}')$ *such that* $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}_{\mathsf{acc}}$*,* $(\mathbb{i}, \mathbb{x}'; \mathbb{w}') \in \mathbf{R}$ *; given global parameters* gp *and an* $(n_1, \ldots, n_\mu)$*-tree of accepting transcripts and the corresponding folded tuples* $(\mathbb{i}, \mathbb{x}^*; \mathbb{w}^*)$*. The tree comprises of* n_1 *transcripts (and the corresponding index instance witness tuples) with fresh randomness in the Verifier's first message ; and for each such transcript,* n_2 *transcripts (and the corresponding index instance witness tuples) with fresh randomness in the Verifier's second message ; and so on, for a total of* $\prod_{i=1}^{\mu} n_i$ *leaves bounded by* $\mathrm{poly}(\lambda)$*.*

4 An IOP and a folding scheme for checking that all rows in a matrix are elementary basis vectors

It is a now standard observation [ZBK+22, ZGK+22] that a tuple $((\mathbb{F}, N, m), (\mathbf{a}, \mathbf{t}))$ is in $\mathbf{R}_{\mathsf{Look}}$ if and only if there exists a matrix $M \in \mathbb{F}^{m \times N}$ such that:

- $M \cdot \mathbf{t}^{\mathsf{T}} = \mathbf{a}^{\mathsf{T}}$,
- The rows of M are vectors in the standard basis of $\mathbb{F}^N$.

One way to build a folding scheme for lookups is to build a folding scheme for the second condition above. Indeed, if as a result of folding the second condition we combine lookup matrices as $M_1 + \alpha M_2$, then the first condition can also be combined as $(M_1 + \alpha M_2) \cdot \mathbf{t}^\mathsf{T} = (\mathbf{a}_1 + \alpha \mathbf{a}_2)^\mathsf{T}$. By defining an adequate accumulated relation, we can ensure that this is also the case when folding a lookup matrix into an accumulated matrix, one that is the result of potentially many foldings. We let $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ be an extractable multilinear PCS. For simplicity, we abbreviate Commit as cm. We want to show that a matrix M seen as a $\log(mN)$-variate multilinear polynomial is in $\mathbf{R}_{\mathsf{elem}}$:

$$\mathbf{R}_{\mathsf{elem}} := \left\{ ((\mathbb{F}, N, m), \overline{M}; M) \;\middle|\; \begin{array}{l} m \leq N < \mathrm{char}(\mathbb{F}), \\ M \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_{\log(m)}, Y_1, \ldots, Y_{\log(N)}], \\ \overline{M} = \mathsf{cm}(M),\ M \circ M = M,\ M \cdot \mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T} \end{array} \right\} \tag{6}$$

The notations $M \circ M = M$ and $M \cdot \mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T}$ are shorthand for:

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathbb{B}^{\log(m)+\log(N)},\ M(\mathbf{x}, \mathbf{y})^2 - M(\mathbf{x}, \mathbf{y}) = 0 \tag{7}$$

$$\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{X}, \mathbf{y}) \equiv 1 \tag{8}$$

These two properties are exactly what we need in order to enforce the fact that each row of M has exactly one 1, and zeros otherwise. Clearly if that is the case, then Eq. (7) and Eq. (8) hold. The converse is also true:

Lemma 4.1. *Suppose $N < \mathrm{char}(\mathbb{F})$. If Eq. (7) and Eq. (8) hold, then for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$, there exists a unique $\mathbf{y}_\mathbf{x} \in \mathbb{B}^{\log(N)}$ such that $M(\mathbf{x}, \mathbf{y}_\mathbf{x}) = 1$, and $M(\mathbf{x}, \mathbf{y}) = 0$ for all $\mathbf{y} \in \mathbb{B}^{\log(N)}$ with $\mathbf{y} \neq \mathbf{y}_\mathbf{x}$.*

Proof. Eq. (7) implies that for all $(\mathbf{x}, \mathbf{y}) \in \mathbb{B}^{\log(m)+\log(N)}$, $M(\mathbf{x}, \mathbf{y}) = 0$ or $M(\mathbf{x}, \mathbf{y}) = 1$. Then Eq. (8) implies that for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$, $\sum_\mathbf{y} M(\mathbf{x}, \mathbf{y}) = 1$. But since $N < \mathrm{char}(\mathbb{F})$, we have that:

$$1 = \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{x}, \mathbf{y}) = |\{\mathbf{y} \in \mathbb{B}^{\log(N)} \mid M(\mathbf{x}, \mathbf{y}) = 1\}|$$

□

The PIOP we describe for the relation $\mathbf{R}_{\mathsf{elem}}$[5] is as follows:

1. The Prover sends an oracle $[[M]]$ to a multilinear polynomial supposedly in $\mathbf{R}_{\mathsf{elem}}$.
2. The Verifier samples uniform random elements $\boldsymbol{\beta} \in \mathbb{F}^{\log(mN)}$, $\mathbf{r} \in \mathbb{F}^{\log(m)}$.

[5] This is slightly abusing notation: the PIOP we describe is for the oracle relation where we replace commitments in $\mathbf{R}_{\mathsf{elem}}$ with oracles.

3. The Prover and Verifier engage in two sumcheck protocols:

$$\sum_{\mathbf{x}\in\mathbb{B}^{\log(m)}} \sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} (M(\mathbf{x},\mathbf{y})^2 - M(\mathbf{x},\mathbf{y})) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta};\mathbf{x},\mathbf{y}) = 0$$
$$\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M(\mathbf{r},\mathbf{y}) = 1$$

4. During these sumchecks, the Verifier runs the sumcheck Verifier on each of the sumchecks. It accepts if the sumcheck Verifier accepts both executions of the sumcheck, and rejects otherwise. Note that in particular, the Verifier queries evaluations of the form $M(\mathbf{r}_1,\mathbf{r}_2)$, $M(\mathbf{r},\mathbf{r}_3)$, for some random elements $\mathbf{r}_1 \in \mathbb{F}^{\log(m)}$ and $\mathbf{r}_2,\mathbf{r}_3 \in \mathbb{F}^{\log(N)}$ determined during the sumchecks. The Verifier can evaluate $\widetilde{\mathsf{eq}}(\boldsymbol{\beta};\mathbf{r}_1,\mathbf{r}_2)$ on its own.

Lemma 4.2. *The above protocol is a perfectly complete and knowledge sound PIOP for the relation* $\mathbf{R}_{\mathsf{elem}}$*, as long as* $(4\log(mN)+1)/|\mathbb{F}| = \mathrm{negl}(\lambda)$.

Proof. Appendix C.1 □

4.1 A folding scheme for $\mathbf{R}_{\mathsf{elem}}$

We can now construct a folding scheme for $\mathbf{R}_{\mathsf{elem}}$ with ideas similar to Nova [KST22]. Note however, that since our statement is particularly simple we are able to perform a simplification of the cross-term that appears when folding. We let $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ be a succinct, binding, extractable, additively homomorphic multilinear PCS. For simplicity, we abbreviate Commit as cm. We define:

$$\mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}} := \left\{ \begin{pmatrix} (\mathbb{F},N,m), \\ (\overline{M},\overline{E},\mu); \\ (M,E) \end{pmatrix} \;\middle|\; \begin{array}{l} m \le N < \mathrm{char}(\mathbb{F}), \\ M,E \in \mathbb{F}^{\le 1}[X_1,\ldots,X_{\log(m)},Y_1,\ldots,Y_{\log(N)}], \mu \in \mathbb{F}, \\ \overline{M} = \mathsf{cm}(M),\ \overline{E} = \mathsf{cm}(E) \\ M \circ M = M + E,\ M \cdot \mathbf{1}^{\mathsf{T}} = (1+\mu)\cdot \mathbf{1}^{\mathsf{T}} \end{array} \right\}$$

and we now describe a $(\mathbf{R}_{\mathsf{elem}} \to \mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}})$-folding scheme, which we call $\mathscr{P}_1 = (\mathsf{P}_1, \mathsf{V}_1)$. We describe it as an interactive protocol, but it can be made non-interactive with the Fiat-Shamir heuristic [FS86].

Input. The protocol input is $(\overline{M_{\mathsf{acc}}}, \overline{E_{\mathsf{acc}}}, \overline{M}, \mu; M_{\mathsf{acc}}, M)$[6].

1. P_1 computes and sends the commitment $\overline{T}$ to the cross term: $T = 2(M_{\mathsf{acc}} \circ M) - M$.

[6]We use the convention that both Prover and Verifier get what is before the semicolon, and only the Prover gets what is after.

2. V_1 sends uniformly random $\alpha \in \mathbb{F}$.

3. P_1 and V_1 output:

$$\overline{M_{\mathsf{acc}}} \leftarrow \overline{M_{\mathsf{acc}}} + \alpha \cdot \overline{M}, \quad \overline{E_{\mathsf{acc}}} \leftarrow \overline{E_{\mathsf{acc}}} + \alpha \cdot \overline{T} + \alpha^2 \cdot \overline{M}, \quad \mu \leftarrow \mu + \alpha.$$

4. P_1 outputs: $M_{\mathsf{acc}} \leftarrow M_{\mathsf{acc}} + \alpha \cdot M, \quad E_{\mathsf{acc}} \leftarrow E_{\mathsf{acc}} + \alpha \cdot T + \alpha^2 \cdot M.$

Lemma 4.3. *Protocol $\mathscr{P}_1$ is a $(\mathbf{R}_{\mathsf{elem}} \to \mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}})$-folding scheme which is perfectly complete, and knowledge sound.*

Proof. Appendix C.2 □

Costs We see that in protocol $\mathscr{P}_1$:

- P_1 performs m field doublings, and $8m + 1$ field operations. It also performs 3 group operations, and 3 exponentiations of group elements to the α power. The Prover needs to additionally commit to to m field elements (the cross term T).
- V_1 performs 3 group exponentiations to the α power, 3 group operations, and one field addition.

4.2 A protocol for proving accumulated instances

To prove a statement of the form $((N, m), \overline{M}, \overline{E}, \mu; M, E) \in \mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}}$, two sumchecks of the form:

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} (M(\mathbf{x}, \mathbf{y})^2 - M(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}, \mathbf{y})) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}; \mathbf{x}, \mathbf{y}) = 0$$
$$\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{r}, \mathbf{y}) = 1 + \mu$$

are performed, for random $\boldsymbol{\beta}, \mathbf{r}$ chosen by the Verifier. In particular in the end, the Verifier needs to verify evaluations of the form $M(\mathbf{r}_1, \mathbf{r}_2), E(\mathbf{r}_1, \mathbf{r}_2), M(\mathbf{r}, \mathbf{r}_3)$ for some random $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ determined during the sumchecks. Note that by the updating procedure of M^{acc} and E^{acc} in $\mathscr{P}_1$, if we have folded $M_0, \ldots, M_{n_f}$ (where initially $M^{\mathsf{acc}} = M_0$), then it holds that at the end of these n_f foldings, the support[7] of M^{acc} and of E^{acc} is included in the union of the supports of the M_i (equality is possible), in particular they are both at most $n_f \cdot m$-sparse. Hence, we may consider that M, E are s-sparse where $s \leq \min\{n_f \cdot m, mN\}$. Provided that we have all the evaluations of $(M(\mathbf{x}, \mathbf{y})^2 - M(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}, \mathbf{y}))$ and of $M(\mathbf{r}, \mathbf{y})$

[7] The support of f is the set of $\mathbf{x}$ such that $f(\mathbf{x}) \neq 0$

over their respective hypercubes, [DT24] shows that we can perform the first sumcheck in $2mN+O(\sqrt{mN})$ field multiplications and the second in N field multiplications. Computing the evaluations of $(M(\mathbf{x},\mathbf{y})^2 - M(\mathbf{x},\mathbf{y}) - E(\mathbf{x},\mathbf{y}))$ over $\mathbb{B}^{\log(mN)}$ takes s field multiplications. Computing the evaluations of $M(\mathbf{r},\mathbf{y})$ over $\mathbb{B}^{\log(N)}$ can be done in at most $s+m$ field multiplications, which can be seen by first computing the table of values of $\widetilde{\mathsf{eq}}(\mathbf{x};\mathbf{r})$ for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$ in m multiplications and writing M in the multilinear Lagrange basis.

Lemma 4.4. *The protocol above is a perfectly complete, knowledge sound PIOP for the relation* $\mathbf{R}_{\mathsf{elem}}^{\mathsf{acc}}$. *Suppose* M *and* E *are* s*-sparse (for* $s \leq mN$*), then in this PIOP the Prover performs at most* $2mN + N + 2s + m + O(\sqrt{mN})$ *field multiplications, and the Verifier performs* $O(\log(mN))$ *field operations. The Verifier makes three oracle queries* $M(\mathbf{r}_1,\mathbf{r}_2), E(\mathbf{r}_1,\mathbf{r}_2),\ M(\mathbf{r},\mathbf{r}_3)$.

Proof. The proof is very similar to Lemma 4.2. □

5 FLI: Folding Lookup Instances

In this section, we describe a method to fold a lookup into a certain relaxed lookup relations that we describe. Starting from the fact that a statement of the form $((\mathbb{F},N,m),(\mathbf{a},\mathbf{t})) \in \mathbf{R}_{\mathsf{Look}}$ can be expressed as an equation $M\cdot\mathbf{t}^{\mathsf{T}} = \mathbf{a}^{\mathsf{T}}$ where the rows of M are elementary basis vectors, we then use the folding scheme we have developed in Section 4. As we mentioned, the random elements used to combine the claims that the matrices M have the correct form will also allow us to fold the claim that $M\cdot\mathbf{t}^{\mathsf{T}} = \mathbf{a}^{\mathsf{T}}$. We let $\mathsf{PC} = (\mathsf{Setup},\mathsf{Commit},\mathsf{Open},\mathsf{Eval})$ be a succinct, binding, extractable, additively homomorphic multilinear PCS. For simplicity, we abbreviate Commit as cm. Start by defining the relaxed relation:

$$\mathbf{R}_1^{\mathsf{acc}} := \left\{ \begin{pmatrix} (\mathbb{F},N,m,\mathbf{t}),\\ (\overline{\mathbf{a}},\overline{M},\overline{E},\mu);\\ (\mathbf{a},M,E) \end{pmatrix} \;\middle|\; \begin{array}{c} m \leq N < \mathrm{char}(\mathbb{F}),\\ M,E \in \mathbb{F}^{\leq 1}[X_1,\ldots,X_{\log(m)},Y_1,\ldots,Y_{\log(N)}],\\ \mathbf{t} \in \mathbb{F}^{\leq 1}[Y_1,\ldots,Y_{\log(N)}],\\ \mathbf{a} \in \mathbb{F}^{\leq 1}[X_1,\ldots,X_{\log(m)}], \mu \in \mathbb{F},\\ \overline{M} = \mathsf{cm}(M),\ \overline{E} = \mathsf{cm}(E), \overline{\mathbf{a}} = \mathsf{cm}(\mathbf{a}),\\ \sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M(\mathbf{X},\mathbf{y})\cdot\mathbf{t}(\mathbf{y}) = \mathbf{a}(\mathbf{X}),\\ M\circ M = M + E,\ M\cdot\mathbf{1}^{\mathsf{T}} = (1+\mu)\cdot\mathbf{1}^{\mathsf{T}} \end{array} \right\}$$

This is essentially the accumulated relation in the previous section augmented with the lookup constraint $M\cdot\mathbf{t}^{\mathsf{T}} = \mathbf{a}^{\mathsf{T}}$. By using our folding scheme $\mathscr{P}_1$ for $\mathbf{R}_{\mathsf{elem}}$, we construct folding schemes for the relation $\mathbf{R}_{\mathsf{CmMAILook}}$ into $\mathbf{R}_1^{\mathsf{acc}}$.

5.1 FLI: a $(\mathbf{R}_{\mathsf{CmMAIlook}} \to \mathbf{R}_1^{\mathsf{acc}})$-folding scheme

The folding scheme is described as an interactive protocol, but it can be made non-interactive with the Fiat-Shamir heuristic [FS86]. We denote this folding scheme by $\mathscr{F}_1 = (\tilde{\mathsf{P}}_1, \tilde{\mathsf{V}}_1)$.

Input. The protocol input is[8]

$$((N, m), \mathbf{t}, \overline{\mathbf{a}_{\mathsf{acc}}}, \overline{M_{\mathsf{acc}}}, \overline{E_{\mathsf{acc}}}, \mu, \overline{M}, \bar{\mathbf{a}}; \mathbf{a}_{\mathsf{acc}}, M_{\mathsf{acc}}, E_{\mathsf{acc}}, \mathbf{a}, M).$$

1. $\tilde{\mathsf{P}}_1$ and $\tilde{\mathsf{V}}_1$ follow protocol $\mathscr{P}_1$ with input

$$((N, m), \overline{M_{\mathsf{acc}}}, \overline{E_{\mathsf{acc}}}, \overline{M}, \mu; M_{\mathsf{acc}}, E_{\mathsf{acc}}, M).$$

At the end of $\mathscr{P}_1$, $\tilde{\mathsf{P}}_1$ and $\tilde{\mathsf{V}}_1$ output:

$$\overline{M_{\mathsf{acc}}} \leftarrow \overline{M_{\mathsf{acc}}} + \alpha \cdot \overline{M}, \quad \overline{E_{\mathsf{acc}}} \leftarrow \overline{E_{\mathsf{acc}}} + \alpha \cdot \overline{T} + \alpha^2 \cdot \overline{M}, \quad \mu \leftarrow \mu + \alpha$$

while $\tilde{\mathsf{P}}_1$ outputs:

$$M_{\mathsf{acc}} \leftarrow M_{\mathsf{acc}} + \alpha \cdot M, \quad E_{\mathsf{acc}} \leftarrow E_{\mathsf{acc}} + \alpha \cdot T + \alpha^2 M$$

for some random element $\alpha \in \mathbb{F}$ determined during the course of $\mathscr{P}_1$, and $T := 2(M_{\mathsf{acc}} \circ M) - M$.

2. $\tilde{\mathsf{P}}_1$ and $\tilde{\mathsf{V}}_1$ output: $\overline{\mathbf{a}_{\mathsf{acc}}} \leftarrow \overline{\mathbf{a}_{\mathsf{acc}}} + \alpha \cdot \bar{\mathbf{a}}$; and $\tilde{\mathsf{P}}_1$ outputs: $\mathbf{a}_{\mathsf{acc}} \leftarrow \mathbf{a}_{\mathsf{acc}} + \alpha \cdot \mathbf{a}$.

Lemma 5.1. *Protocol $\mathscr{F}_1$ is a $(\mathbf{R}_{\mathsf{CmMAIlook}} \to \mathbf{R}_1^{\mathsf{acc}})$-folding scheme that is perfectly complete, and knowledge sound.*

Proof. Appendix C.3 □

Costs We see that the Prover and Verifier work in $\mathscr{F}_1$ is almost the same as in $\mathscr{P}_1$. The additional work is: one group exponentiation, one group multiplication for both the Prover and Verifier. Further, the Prover makes $2m$ extra field operations.

[8] Still with the convention that both Prover and Verifier get what is before the semicolon, and only the Prover gets what is after.

5.2 A protocol for proving accumulated instances.

To prove a statement of the form$((N,m),\mathbf{t},\overline{\mathbf{a}},\overline{M},\overline{E},\mu;\mathbf{a},M,E)\in\mathbf{R}_1^{\mathsf{acc}}$, we use the same protocol that proves accumulated instances for $\mathscr{P}_1$ (Section 4.2), together with an additional sumcheck of the form:

$$\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M(\mathbf{r},\mathbf{y})\cdot\mathbf{t}(\mathbf{y})=\mathbf{a}(\mathbf{r})$$

for some random $\mathbf{r}$. Again, we may assume M,E are s-sparse with $s\leq\min\{n_f\cdot m, mN\}$ (n_f the number of folding steps, see Section 4.2). Using [DT24], if we have all evaluations of $M(\mathbf{r},\mathbf{y})\mathbf{t}(\mathbf{y})$ over $\mathbb{B}^{\log(N)}$, this sumcheck costs $5N$ field multiplications for the Prover. Computing all evaluations of $M(\mathbf{r},\mathbf{y})\mathbf{t}(\mathbf{y})$ can be done in $2s+m$ field multiplications. We can also save costs by batching this sumcheck with the sumcheck of the form $\sum_{\mathbf{y}} M(\mathbf{r}',\mathbf{y})=(1+\mu)$ from the protocol that proves accumulated instances for $\mathscr{P}_1$. The Verifier samples a single random element $\mathbf{r}\in\mathbb{F}^{\log(m)}$ for both of these sumchecks, and an additional combination random element $\gamma\in\mathbb{F}$. All in all, the protocol performs the two following sumchecks:

$$\sum_{\mathbf{x}\in\mathbb{B}^{\log(m)}}\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}}(M(\mathbf{x},\mathbf{y})^2-M(\mathbf{x},\mathbf{y})-E(\mathbf{x},\mathbf{y}))\cdot\widetilde{\mathsf{eq}}(\beta;\mathbf{x},\mathbf{y})=0$$

$$\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M(\mathbf{r},\mathbf{y})+\gamma\cdot M(\mathbf{r},\mathbf{y})\cdot\mathbf{t}(\mathbf{y})=1+\mu+\gamma\cdot\mathbf{a}(\mathbf{r}),$$

and reduces to the evaluations $M(\mathbf{r}_1,\mathbf{r}_2),E(\mathbf{r}_1,\mathbf{r}_2),M(\mathbf{r},\mathbf{r}_3),\mathbf{a}(\mathbf{r})$ for the random elements $\mathbf{r}_1\in\mathbb{F}^{\log(m)}$ and $\mathbf{r}_2,\mathbf{r}_3\in\mathbb{F}^{\log(N)}$ determined during the sumchecks.

Lemma 5.2. *The protocol above is a perfectly complete and knowledge sound PIOP for the relation $\mathbf{R}_1^{\mathsf{acc}}$. Suppose M and E are s-sparse (for $s\leq mN$), then in this PIOP the Prover performs*

$$2mN+N+3s+m+O(\sqrt{mN})$$

field multiplications, and the Verifier performs $O(\log(mN))$ field operations. The Verifier makes four oracle queries $M(\mathbf{r}_1,\mathbf{r}_2),E(\mathbf{r}_1,\mathbf{r}_2),M(\mathbf{r},\mathbf{r}_3),\mathbf{a}(\mathbf{r})$.

5.3 Extending SOS decompositions for folding

In Lasso [STW24], when $\mathbf{t}$ is SOS-decomposable (into $\alpha=k\cdot c$ tables $\mathbf{t}_1,\ldots,\mathbf{t}_\alpha$ and polynomial g, see Definition 3.6), one needs to verify that for all $\mathbf{x}\in\mathbb{B}^{\log(m)}$ the following condition holds:

$$\mathbf{a}(\mathbf{x})=g(\mathbf{E}_1(\mathbf{x}),\ldots,\mathbf{E}_\alpha(\mathbf{x}))\wedge\exists\mathbf{y}=(\mathbf{y}_1,\ldots,\mathbf{y}_c)\in\left(\mathbb{B}^{\log(N)/c}\right)^c,\ \mathbf{E}_i(\mathbf{x})=\mathbf{t}_i(\mathbf{y}_{\lceil i/k\rceil})$$

Where the $\mathbf{E}_i$ are vectors of length m. The first of these conditions is verified with a sumcheck, and Lasso uses an offline memory-checking technique to certify that the second

condition holds. Note however, that since the $\mathbf{E}_i$ are included in the $\mathbf{t}_i$, their entries could a priori be arbitrary elements of $\mathbb{F}$ [9]. As is remarked in Lasso, this can represent a significant overhead when committing to the vectors $\mathbf{E}_i$ by using curve-based schemes.

As we remarked when discussing Eq. (4), we can express the same conditions in a different way. When $\mathbf{t}$ is SOS-decomposable, the fact that for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$ there exists $\mathbf{y} \in \mathbb{B}^{\log(N)}$ such that $\mathbf{a}(\mathbf{x}) = \mathbf{t}(\mathbf{y})$ can be expressed as:

$$\forall \mathbf{x} \in \mathbb{B}^{\log(m)},\ \exists \mathbf{y} \in \mathbb{B}^{n},\ \mathbf{a}(\mathbf{x}) = \mathbf{t}(\mathbf{y}) = g(\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_k(\mathbf{y}_1), \mathbf{t}_{k+1}(\mathbf{y}_2), \ldots, \mathbf{t}_\alpha(\mathbf{y}_c)) \tag{9}$$

So we can think of Eq. (9) as needing to point, for each $\mathbf{x} \in \mathbb{B}^{\log(m)}$, to the correct indices of the tables $\mathbf{t}_1, \ldots, \mathbf{t}_\alpha$ that make the equation hold. Note that the $\mathbf{t}_i$ have size $N^{1/c}$, so we know that we can point to the correct indices for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$ by using matrices of size $m \times N^{1/c}$ that have elementary basis vectors as rows. Therefore Eq. (9) is equivalent to:

$$\forall \mathbf{x} \in \mathbb{B}^{\log(m)},\ \mathbf{a}(\mathbf{x}) = g\left(M_1(\mathbf{x})\mathbf{t}_1, \ldots, M_1(\mathbf{x})\mathbf{t}_k, M_2(\mathbf{x})\mathbf{t}_{k+1}, \ldots, M_c(\mathbf{x})\mathbf{t}_\alpha\right) \tag{10}$$

for some matrices $M_1, \ldots, M_c$ of size $m \times N^{1/c}$ which have the special form we have been talking about: *each of their rows is a vector in the standard basis of* $\mathbb{F}^{N^{1/c}}$. Importantly, committing to the M_i with curve-based commitment schemes consists only in group *operations*. As we mentioned, one should think as the matrices M_i (for $i \in [c]$) as pointing to the correct entries of $\mathbf{t}_{(i-1)k+1}, \ldots, \mathbf{t}_{ik}$ such that Eq. (9) holds. For simplicity we use the shorthand:

$$\forall \mathbf{x} \in \mathbb{B}^{\log(m)}, \forall i \in [c],\ (i-1)k + 1 \leq j \leq ik,\ M_i(\mathbf{x})\mathbf{t}_j := \sum_{\mathbf{y} \in \mathbb{B}^n} M_i(\mathbf{x}, \mathbf{y}) \cdot \mathbf{t}_j(\mathbf{y}) \tag{11}$$

Note that the $M_i(\mathbf{X})\mathbf{t}_j$ are $\log(m)$-variate multilinear polynomials whose evaluations over the hypercube can be computed by simply selecting entries from $\mathbf{t}_j$. We can verify Eq. (10) with a sumcheck of the form:

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} \left(\mathbf{a}(\mathbf{x}) - g(M_1(\mathbf{x})\mathbf{t}_1, \ldots, M_1(\mathbf{x})\mathbf{t}_k, M_2(\mathbf{x})\mathbf{t}_{k+1}, \ldots, M_c(\mathbf{x})\mathbf{t}_\alpha)\right) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}; \mathbf{x}) = 0$$

for a random $\boldsymbol{\beta} \in \mathbb{F}^{\log(m)}$, which will reduce to claims of the form $\mathbf{a}(\mathbf{r}) = d, M_i(\mathbf{r})\mathbf{t}_j = v_{i,j}$ for some random $\mathbf{r} \in \mathbb{F}^{\log(m)}$ and $d, v_{i,j} \in \mathbb{F}$ (for $i \in [c]$ and $(i-1)k + 1 \leq j \leq ik$). This will allow us to use our folding scheme for the relation $\mathbf{R}_{\mathsf{CmMAILook}}$ to fold these claims about the evaluations of the $M_i(\mathbf{r})\mathbf{t}_j = v_{i,j}$, as per the following remark.

[9] This is mitigated by the fact that in tables that arise in practice, for example in the RISC-V instruction set, the entries in the tables appearing in the SOS decompositions are relatively small.

Remark 5.3 (The $M \cdot \mathbf{t}^\mathsf{T} = \mathbf{a}^\mathsf{T}$ condition)**.** In Section 5.1 we described a folding scheme for the relation $\mathbf{R}_{\mathsf{CmMAlLook}}$, in which the condition relating M, $\mathbf{t}$ and $\mathbf{a}$ is:

$$\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}(\mathbf{X})$$

We could equally have described the folding scheme for a "randomized" version of this condition, of the form:

$$\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M(\mathbf{r}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}(r)$$

for some random $\mathbf{r} \in \mathbb{F}^{\log(m)}$. The two folding schemes we have described translate mutatis mutandis to this setting, *provided that the random evaluation vector r is the same for the accumulated claim and the new claim.* We do not write this again as it is quite literally the same folding scheme. The protocol that proves accumulated instances of the non-randomized versions of our folding schemes starts precisely by randomizing the condition $\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M^{\mathsf{acc}}(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}^{\mathsf{acc}}(\mathbf{X})$ as $\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M^{\mathsf{acc}}(\mathbf{r}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}^{\mathsf{acc}}(\mathbf{r})$. We still refer to the folding scheme for this randomized condition $M \cdot \mathbf{t}^\mathsf{T} = \mathbf{a}^\mathsf{T}$ as $\mathscr{F}_1$. The soundness loss of this randomization is the probability that the randomized condition holds while the condition on polynomials does not. This has probability at most $\log(m)/|\mathbb{F}|$ by Schwartz-Zippel, since all polynomials are multilinear. This will be useful in the next section, when we use SOS decompositions in conjunction with the folding schemes we have constructed.

5.4 FLI + SOS

In this section, we combine SOS decompositions with FLI using our remarks from Section 5.3: by modifying the SOS decomposition step from the Lasso paper [STW24] we make it such that the Prover only needs to commit to sparse binary matrices. Recall from the previous section that when we are looking up $\mathbf{a}$ into an SOS-decomposable table $\mathbf{t}$ we can express the lookup condition as a sumcheck in $\log(m)$ variables:

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} (\mathbf{a}(\mathbf{x}) - g(M_1(\mathbf{x})\mathbf{t}_1, \ldots, M_1(\mathbf{x})\mathbf{t}_k, M_2(\mathbf{x})\mathbf{t}_{k+1}, \ldots M_c(\mathbf{x})\mathbf{t}_\alpha)) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}; \mathbf{x}) = 0$$

for matrices $M_1, \ldots, M_c$ of size $m \times N^{1/c}$ in $\mathbf{R}_{\mathsf{elem}}$, and a random $\boldsymbol{\beta} \in \mathbb{F}^{\log(m)}$. The sumcheck reduces to claims of the form $\mathbf{a}(\mathbf{r}) = d, M_i(\mathbf{r})\mathbf{t}_j = v_{i,j}$ for some random $\mathbf{r} \in \mathbb{F}^{\log(m)}$ and $d, v_{i,j} \in \mathbb{F}$. We can now fold the claims about the matrices being in $\mathbf{R}_{\mathsf{elem}}$ using Protocol $\mathscr{P}_1$, and in parallel fold the claims about the evaluations. The only important detail (see Remark 5.3) is that the random evaluation point is the same, which we can always enforce with a technique we call the *point-shifting sumcheck*. This is a quite standard technique (see for example [KS24]), that allows to reduce the evaluation of two (or more)

multilinear polynomials at different point to evaluations at the same point. It exploits that for any multilinear polynomial $f \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_s]$ and any $\mathbf{r} \in \mathbb{F}^s$, it holds that $f(\mathbf{r}) = \sum_{\mathbf{x} \in \mathbb{B}^s} f(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \mathbf{r})$ (see Eq. (5)). Say we have two multilinear polynomials f, g in s variables with purported evaluations $f(\mathbf{r}_1) = c, g(\mathbf{r}_2) = d$ respectively (for some $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{F}^s$), we may apply the sumcheck protocol to certify that the following holds:

$$\sum_{\mathbf{x} \in \mathbb{B}^s} f(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \mathbf{r}_1) + \gamma \cdot g(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \mathbf{r}_2) = c + \gamma \cdot d$$

for a random uniform $\gamma \in \mathbb{F}$. At the end of this sumcheck, the Prover needs to reveal $f(\mathbf{r}), g(\mathbf{r})$ and $\widetilde{\mathsf{eq}}(\mathbf{r}; \mathbf{r}_1), \widetilde{\mathsf{eq}}(\mathbf{r}, \mathbf{r}_2)$. In this way, we now have reduced the statements that $f(\mathbf{r}_1) = c, g(\mathbf{r}_2) = d$ to an evaluation of f and g at the same point.

Formally, we will obtain a folding scheme from the committed matrix lookup SOS relation:

$$\mathbf{R}^{\mathsf{SOS}} := \left\{ \begin{pmatrix} (\mathbb{F}, N, m, \mathbf{t}, \alpha, \\ c, k, \mathbf{t}_1, \ldots, \mathbf{t}_\alpha, g), \\ (\overline{M_1}, \ldots, \overline{M_c}, \overline{\mathbf{a}}); \\ (M_1, \ldots, M_c, \mathbf{a}) \end{pmatrix} \;\middle|\; \begin{array}{l} N < \mathrm{char}(\mathbb{F}),\ \alpha = k \cdot c \in \mathbb{N}, \\ \forall i \in [c],\ M_i \in \mathbb{F}^{\leq 1}[\mathbf{X}_{[\log(m)]}, \mathbf{Y}_{[\log(N)/c]}], \\ \mathbf{a} \in \mathbb{F}^{\leq 1}[\mathbf{X}_{[\log(m)]}],\ \mathbf{t} \in \mathbb{F}^{\leq 1}[\mathbf{Y}_{[\log(N)/c]}], \\ \mathbf{t}_1, \ldots, \mathbf{t}_\alpha \in \mathbb{F}^{\leq 1}[\mathbf{Y}_{[\log(N)/c]}], \\ \forall i \in [c],\ \overline{M_i} = \mathsf{cm}(M_i),\ \overline{\mathbf{a}} = \mathsf{cm}(\mathbf{a}), \\ \forall \mathbf{y} \in \mathbb{B}^{\log(N)},\ \mathbf{t}(\mathbf{y}) = g(\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_\alpha(\mathbf{y}_c)), \\ \forall \mathbf{x} \in \mathbb{B}^{\log(m)},\ \mathbf{a}(\mathbf{x}) = g(M_1(\mathbf{x})\mathbf{t}_1, \ldots, M_c(\mathbf{x})\mathbf{t}_\alpha), \\ \forall i \in [c],\ M_i \circ M_i = M_i,\ M_i \cdot \mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T} \end{array} \right\}$$

(for brevity $\mathbf{X}_{[\log(m)]} = (X_1, \ldots X_{\log(m)})$, $\mathbf{Y}_{[\log(N)/c]} = (Y_1, \ldots Y_{\log(N)/c})$) into the accumulated relation:

$$\mathbf{R}^{\mathsf{accSOS}} :=$$

$$\left\{ \begin{pmatrix} (\mathbb{F}, N, m, \mathbf{t}, \alpha, \\ \quad c, k, \mathbf{t}_1, \ldots, \mathbf{t}_\alpha, g), \\ (\overline{M}, \overline{M_1}, \ldots, \overline{M_c}, \overline{E}, \overline{\mathbf{a}}, \mathbf{r}, \\ \quad d, \mu, c_1, \ldots, c_\alpha); \\ (M, M_1, \ldots, M_c, E, \mathbf{a}) \end{pmatrix} \middle| \begin{array}{l} N < \mathrm{char}(\mathbb{F}), \alpha = k \cdot c \in \mathbb{N}, \\ \mathbf{r} \in \mathbb{F}^{\log(m)}, d, c_1, \ldots, c_\alpha, \mu \in \mathbb{F}, \\ M, M_1, \ldots, M_c, E \in \mathbb{F}^{\leq 1}[\mathbf{X}_{[\log(m)]}, \mathbf{Y}_{[\log(N)/c]}], \\ \mathbf{a} \in \mathbb{F}^{\leq 1}[\mathbf{X}_{[\log(m)]}],\ \mathbf{t} \in \mathbb{F}^{\leq 1}[\mathbf{Y}_{[\log(N)/c]}], \\ \mathbf{t}_1, \ldots, \mathbf{t}_\alpha \in \mathbb{F}^{\leq 1}[\mathbf{Y}_{[\log(N)/c]}], \\ \overline{M} = \mathsf{cm}(M),\ \overline{E} = \mathsf{cm}(E), \overline{\mathbf{a}} = \mathsf{cm}(\mathbf{a}), \\ \forall i \in [c],\ \overline{M_i} = \mathsf{cm}(M_i), \\ \forall \mathbf{y} \in \mathbb{B}^{\log(N)},\ \mathbf{t}(\mathbf{y}) = g(\mathbf{t}_1(\mathbf{y}_1), \ldots, \mathbf{t}_\alpha(\mathbf{y}_c)), \\ \mathbf{a}(\mathbf{r}) = d, M_1(\mathbf{r})\mathbf{t}_1 = c_1, \ldots, M_c(\mathbf{r})\mathbf{t}_\alpha = c_\alpha, \\ M \circ M = M + E,\ M \cdot \mathbf{1}^\mathsf{T} = (1+\mu) \cdot \mathbf{1}^\mathsf{T} \end{array} \right\}$$

We describe this next as an interactive protocol, but it can be made non-interactive with the Fiat-Shamir heuristic [FS86]. We denote it by $\mathscr{F}_1^{\mathsf{SOS}} = (\mathsf{P}_1^{\mathsf{SOS}}, \mathsf{V}_1^{\mathsf{SOS}})$.

Input. The Protocol input is

$$\begin{pmatrix} (\mathbb{F}, N, m), \alpha, c, k, \mathbf{t}, \mathbf{t}_1, \ldots, \mathbf{t}_\alpha, g, \overline{M^{\mathsf{acc}}}, \overline{M_1^{\mathsf{acc}}}, \ldots, \\ \overline{M_c^{\mathsf{acc}}}, \overline{E^{\mathsf{acc}}}, \overline{\mathbf{a}^{\mathsf{acc}}}, \mathbf{r}^{\mathsf{acc}}, \mu, d^{\mathsf{acc}}, c_1^{\mathsf{acc}}, \ldots, c_\alpha^{\mathsf{acc}}, \overline{M_1}, \ldots, \overline{M_c}, \overline{\mathbf{a}}; \\ M^{\mathsf{acc}}, M_1^{\mathsf{acc}}, \ldots, M_c^{\mathsf{acc}}, E^{\mathsf{acc}}, \mathbf{a}^{\mathsf{acc}}, M_1, \ldots, M_c, \mathbf{a} \end{pmatrix}$$

1. $\mathsf{P}_1^{\mathsf{SOS}}$ and $\mathsf{V}_1^{\mathsf{SOS}}$ engage in a sumcheck of the form:

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} \phi(\mathbf{x}) = \gamma \cdot c_1^{\mathsf{acc}} + \cdots + \gamma^{\alpha+1} \cdot c_\alpha^{\mathsf{acc}} + \gamma^{\alpha+2} \cdot d^{\mathsf{acc}}$$

where

$$\begin{aligned} \phi(\mathbf{x}) =& (\mathbf{a}(\mathbf{x}) - g(M_1(\mathbf{x})\mathbf{t}_1, \ldots, M_c(\mathbf{x})\mathbf{t}_\alpha))) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}, \mathbf{x}) \\ &+ \gamma \cdot M_1^{\mathsf{acc}}(\mathbf{x})\mathbf{t}_1 \cdot \widetilde{\mathsf{eq}}(\mathbf{r}^{\mathsf{acc}}, \mathbf{x}) + \cdots + \gamma^{\alpha+1} \cdot M_c^{\mathsf{acc}}(\mathbf{x})\mathbf{t}_\alpha \cdot \widetilde{\mathsf{eq}}(\mathbf{r}^{\mathsf{acc}}, \mathbf{x}) \\ &+ \gamma^{\alpha+2} \cdot \mathbf{a}^{\mathsf{acc}}(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{r}^{\mathsf{acc}}, \mathbf{x}) \end{aligned}$$

for some random uniform $\boldsymbol{\beta} \in \mathbb{F}^{\log(m)}$ and $\gamma \in \mathbb{F}$ chosen by $\mathsf{V}_1^{\mathsf{SOS}}$. At the end of the sumcheck, $\mathsf{V}_1^{\mathsf{SOS}}$ needs to check a single equation involving the evaluations

$$\begin{aligned} &\widetilde{\mathsf{eq}}(\boldsymbol{\beta}, \mathbf{r}), \widetilde{\mathsf{eq}}(\mathbf{r}^{\mathsf{acc}}, \mathbf{r}), \mathbf{a}(\mathbf{r}), \mathbf{a}^{\mathsf{acc}}(\mathbf{r}), \\ &M_1(\mathbf{r})\mathbf{t}_1, \ldots, M_c(\mathbf{r})\mathbf{t}_\alpha, M_1^{\mathsf{acc}}(\mathbf{r})\mathbf{t}_1, \ldots, M_c^{\mathsf{acc}}(\mathbf{r})\mathbf{t}_\alpha, \end{aligned}$$

where the random challenge $\mathbf{r} \in \mathbb{F}^{\log(m)}$ is determined during the course of the sumcheck. It evaluates the first two by itself.

2. Now, for $i = 1$ to $i = c$:

- $\mathsf{P}_1^{\mathsf{SOS}}$ computes and sends the commitments $\overline{T_i}$ to the cross term:

$$T_i := 2(M^{\mathsf{acc}} \circ M_i) - M_i$$

- $\mathsf{V}_1^{\mathsf{SOS}}$ chooses a uniform random element $\eta_i \in \mathbb{F}$. $\mathsf{P}_1^{\mathsf{SOS}}$ and $\mathsf{V}_1^{\mathsf{SOS}}$ output:

$$\overline{M^{\mathsf{acc}}} \leftarrow \overline{M^{\mathsf{acc}}} + \eta_i \cdot \overline{M_i}, \quad \overline{E^{\mathsf{acc}}} \leftarrow \overline{E^{\mathsf{acc}}} + \eta_i \cdot \overline{T_i} + \eta_i^2 \cdot \overline{M_i}$$
$$\mu \leftarrow \mu + \eta_i, \quad \overline{M_i^{\mathsf{acc}}} \leftarrow \overline{M_i^{\mathsf{acc}}} + \eta_i \cdot \overline{M_i}$$

  and for all $j \in \{k(i-1)+1, \ldots, ki\}$, $c_j^{\mathsf{acc}} \leftarrow M_i^{\mathsf{acc}}(\mathbf{r})\mathbf{t}_j + \eta_i \cdot M_i(\mathbf{r})\mathbf{t}_j$.

- $\mathsf{P}_1^{\mathsf{SOS}}$ outputs:

$$M^{\mathsf{acc}} \leftarrow M^{\mathsf{acc}} + \eta_i \cdot M_i, \quad E^{\mathsf{acc}} \leftarrow E^{\mathsf{acc}} + \eta_i \cdot T_i + \eta_i^2 \cdot M_i$$
$$M_i^{\mathsf{acc}} \leftarrow M_i^{\mathsf{acc}} + \eta_i \cdot M_i$$

3. $\mathsf{V}_1^{\mathsf{SOS}}$ chooses uniform random element $\theta \in \mathbb{F}$. $\mathsf{P}_1^{\mathsf{SOS}}$ and $\mathsf{V}_1^{\mathsf{SOS}}$ output:

$$\overline{\mathbf{a}^{\mathsf{acc}}} \leftarrow \overline{\mathbf{a}^{\mathsf{acc}}} + \theta \cdot \overline{\mathbf{a}}, \quad \mathbf{r}^{\mathsf{acc}} \leftarrow \mathbf{r}, \quad d^{\mathsf{acc}} \leftarrow \mathbf{a}^{\mathsf{acc}}(\mathbf{r}) + \theta \cdot \mathbf{a}(\mathbf{r})$$

4. $\mathsf{P}_1^{\mathsf{SOS}}$ outputs: $\mathbf{a}^{\mathsf{acc}} \leftarrow \mathbf{a}^{\mathsf{acc}} + \theta \cdot \mathbf{a}$

Lemma 5.4. *Protocol $\mathscr{F}_1^{\mathsf{SOS}}$ is a $(\mathbf{R}^{\mathsf{SOS}} \to \mathbf{R}_1^{\mathsf{accSOS}})$-folding scheme that is perfectly complete, and knowledge sound.*

Proof. Appendix C.4 □

5.4.1 Costs

All in all, we see that in protocol $\mathscr{F}_1^{\mathsf{SOS}}$:

- The Prover needs to perform the initial sumcheck which costs $m \cdot (\max(\deg(g), 2)+1) \cdot (6\alpha + |g| + 12)$ field operations using the formula from [Hab22]. The Prover needs to perform $4c+1$ group exponentiations, and $4c+1$ group additions. The Prover also needs to perform $O(\alpha m)$ field operations, where the constant is small.
- The Verifier needs to perform $O(\alpha \log(m))$ field operations in the sumcheck, $4c+1$ group exponentiations, and $4c+1$ group additions. It also needs to perform $2\alpha + 2$ field operations.

5.4.2 Proving accumulated instances

To prove accumulated instances, we need to verify the following:

$$\mathbf{a}(\mathbf{r}) = d,\ M_1(\mathbf{r})\mathbf{t}_1 = c_1, \ldots, M_c(\mathbf{r})\mathbf{t}_\alpha = c_\alpha$$
$$M \circ M = M + E,\ M \cdot \mathbf{1}^\mathsf{T} = (1+\mu)\cdot \mathbf{1}^\mathsf{T}$$

The first claim is proved with an evaluation query to $\mathbf{a}$. The second and last claim are solved with a single sumcheck of the form:

$$\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)/c}} M(\mathbf{r}',\mathbf{y}) + \delta\cdot M_1(\mathbf{r},\mathbf{y})\cdot\mathbf{t}_1(\mathbf{y}) + \delta^\alpha\cdot M_c(\mathbf{r},\mathbf{y})\cdot\mathbf{t}_\alpha(\mathbf{y}) = (1+\mu) + \delta\cdot c_1 + \cdots + \delta^\alpha\cdot c_\alpha$$

for a randomly sampled $\delta \in \mathbb{F}$ and $\mathbf{r}' \in \mathbb{F}^{\log(m)}$. Similarly to Section 4.2 and Section 5.2, we may assume that the M, M_i, E are s-sparse ($s \le mN^{1/c}$). To apply the techniques in [DT24], we need to compute the evaluations $M(\mathbf{r}',\mathbf{y})$ and $\delta\cdot M_1(\mathbf{r},\mathbf{y})\cdot\mathbf{t}_1(\mathbf{y}), \ldots, \delta^\alpha\cdot M_c(\mathbf{r},\mathbf{y})\cdot\mathbf{t}_\alpha(\mathbf{y})$ over $\mathbb{B}^{\log(N)/c}$. This can be done in no more than $s+m$ field multiplications for $M'(\mathbf{r}',\mathbf{y})$, and $3\alpha s + m$ field multiplications for all the other products. Hence, the Prover in this sumcheck does no more than $(5\alpha+1)N^{1/c} + (3\alpha+1)s + 2m$ field multiplications. At the end of the sumcheck the Prover needs to reveal evaluations $M_1(\mathbf{r},\mathbf{r}_1), \ldots, M_c(\mathbf{r},\mathbf{r}_1), M(\mathbf{r}',\mathbf{r}_1), \mathbf{t}_1(\mathbf{r}_1), \ldots, \mathbf{t}_\alpha(\mathbf{r}_1)$ for $\mathbf{r}_1 \in \mathbb{F}^{\log(N)/c}$ determined during the sumcheck. By the SOS assumption the Verifier can evaluate the $\mathbf{t}_i$'s in $O(\alpha\log(N)/c)$ field operations. The remaining condition is also checked with a single sumcheck of the form:

$$\sum_{\mathbf{x}\in\mathbb{B}^{\log(m)}}\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)/c}} ((M(\mathbf{x},\mathbf{y}))^2 - M(\mathbf{x},\mathbf{y}) - E(\mathbf{x},\mathbf{y}))\cdot\widetilde{\mathsf{eq}}(\boldsymbol{\beta};\mathbf{x},\mathbf{y}) = 0$$

for a randomly sampled $\boldsymbol{\beta}$. We already saw that in this sumcheck the Prover performs no more than $2mN^{1/c} + s + O(\sqrt{mN^{1/c}})$ field multiplications. At the end of this sumcheck, the Prover needs to reveal evaluations $M(\mathbf{r}_2,\mathbf{r}_3), E(\mathbf{r}_2,\mathbf{r}_3), \widetilde{\mathsf{eq}}(\boldsymbol{\beta};\mathbf{r}_2,\mathbf{r}_3)$ for $\mathbf{r}_2 \in \mathbb{F}^{\log(m)}, \mathbf{r}_3 \in \mathbb{F}^{\log(N)/c}$ determined during the sumcheck. The Verifier can evaluate $\widetilde{\mathsf{eq}}(\boldsymbol{\beta};\mathbf{r}_2,\mathbf{r}_3)$ in $O(\log(m)+\log(N)/c)$ field operations.

Lemma 5.5. *The protocol Π_1^{accSOS} we just described is a perfectly complete and knowledge sound PIOP for the relation $\mathbf{R}_1^{\mathsf{accSOS}}$. Suppose all M_i, M, E are s-sparse (for $s \le mN^{1/c}$), then in this PIOP the Prover performs no more than:*

$$2mN^{1/c} + (5\alpha+1)N^{1/c} + (3\alpha+2)s + 2m + O(\sqrt{mN^{1/c}})$$

field multiplications, and the Verifier performs $O(\log(mN^{1/c}))$ field operations. The Verifier makes $c+4$ oracle queries $M_1(\mathbf{r},\mathbf{r}_1), \ldots, M_c(\mathbf{r},\mathbf{r}_1)$, $M(\mathbf{r}',\mathbf{r}_1)$, $M(\mathbf{r}_2,\mathbf{r}_3), E(\mathbf{r}_2,\mathbf{r}_3)$, $\mathbf{a}(\mathbf{r})$.

5.5 Using Mova in FLI+SOS

Mova [DGMV24] is a recently published folding scheme for R1CS instances that eliminates the need to commit to the cross-term in Nova [KST22]. Further, and contrary to Hypernova [KS24], it does not use sumchecks and has only 3 rounds of interaction. This makes Mova particularly lightweight, and to the best of our knowledge it is the cheapest folding scheme for R1CS. The concrete Prover and Verifier work of Mova is as follows:

Table 5: Concrete Prover and Verifier costs of Mova [DGMV24]. Here the R1CS matrices are $m \times m$ matrices with $n = \Omega(m)$ non-zero entries, and we assume that the public inputs are a vector of size $\ell < m$. We also assume that the commitment scheme for the witness is curve-based. Prover work takes into account field multiplications (before the $\mathbb{F}$ symbol) and group operations. Verifier work also takes into account field additions.

Prover work	**Verifier work**	**Rounds**
$3n + 12m + 3\log(m)\,\mathbb{F}$ $1\,\mathbb{G}$ exp., $1\,\mathbb{G}$ op.	$2\ell + 7\log(m) + 5\,\mathbb{F}$ $1\,\mathbb{G}$ exp., $1\,\mathbb{G}$ op.	3

One can use Mova as a black box in the construction of FLI+SOS (see Section 5.4). Informally, the relation $\mathbf{R}^{\mathsf{accSOS}}$ should be modified so that the commitment to E is replaced with the supposed evaluation of the MLE of E at a vector of field elements $\tilde{\mathbf{r}} \in \mathbb{F}^{\log(m)}$. The second point in the construction of $\mathscr{F}_1^{\mathsf{SOS}}$ needs to be modified as follows:

2. For $i = 1$ to $i = c$: $\mathsf{P}_1^{\mathsf{SOS}}$ and $\mathsf{V}_1^{\mathsf{SOS}}$ engage in the Mova protocol to combine M^{acc} and M_i and for all $j \in \{k(i-1)+1, \ldots, ki\}$, $c_j^{\mathsf{acc}} \leftarrow M_i^{\mathsf{acc}}(\mathbf{r})\mathbf{t}_j + \eta_i \cdot M_i(\mathbf{r})\mathbf{t}_j$. The randomness used to combine M^{acc} and M_i is used to update the value of μ.

In our case, the R1CS matrices happen to all be the identity (the condition is a relaxation of $M \circ M = M$), there are no public inputs, and the witnesses are of size $mN^{1/c}$. This means that the Prover and Verifier would incur costs dominated by:

- $15cmN^{1/c}$ field multiplications and c group exponentiations for the Prover.
- $O(c\log(m))$ field operations and c group exponentiations for the Verifier.

Further, the folding scheme would have $\log(m) + 3c$ rounds. As we explained, FLI+SOS is particularly efficient when there have not been many folding steps, because in that case the cross-term consists mostly of zeros. This variation where we use Mova in FLI+SOS could be very useful when we know that we will applying the folding step a large number of times. In that case, it is likely that FLI+SOS will eventually operate in the "worst case

scenario" where the cross term is dense. If $15cmN^{1/c}$ is less than the equivalent number of field multiplications needed to compute a commitment to a vector of size m with arbitrary entries in $\mathbb{F}$, then one should use this variation of FLI+SOS.

6 Performance

In this section we discuss performance aspects of FLI+SOS. First, it is clear from Tables 2 and 3 that FLI+SOS has the most efficient folding Prover and Verifier compared to Protostar+SOS and DeepThought+SOS current state-of-the-art schemes. When it comes to proving accumulated instances, FLI+SOS's Prover incurs a cost with a term of the form $2mN^{1/c}$. While this is prohibitive in some parameter settings, we argue that for many regimes of interest, FLI+SOS's concrete costs for proving accumulated instances are comparable to all alternatives. In particular, this is the case when $N^{1/c}$ is small, and m is relatively large. This scenario arises naturally in applications such as continuations in Jolt [AST24, Tha24] (cf. Section 1). All in all, we argue that FLI+SOS can be the best scheme "end-to-end": for folding lookup instances, and proving the resulting accumulated instance.

We consider for example two concrete settings: one where $m = 2^{17}$, $N = 2^{1024}$, $N^{1/c} = 2^8$, $c = 128$, and $\alpha = 2c$; and another where we keep the same values of m, N, but we take the SOS decomposition into smaller tables with $c = 256$ and $\alpha = 2c$. Recall m is the size of the small table, N the size of the big table, which we assume is SOS-decomposable into α tables of size $N^{1/c}$ (see Definition 3.6). This setting resembles some of the parameter choices suggested in [AST24, Tha24]. Further, assume that we perform $n_f = 2^3$ folding steps, which would allow us to prove $m \cdot n_f = 2^{20}$ lookups. Based on these parameters, in Table 6 and Table 7 we use Tables 2 to 4 and 8 to approximate the costs of the folding Prover and the Prover for proving accumulated instances, counted in field multiplications and polynomial openings. Afterward, we briefly discuss how these costs were obtained, and mention possible variations in the schemes that would lead to other costs.

We discuss the setting where m is small and $N^{1/c}$ is relatively large in Appendix B. In such setting, FLI essentially only requires committing to binary vectors, due to the average case commitment cost of FLI (Tables 2 and 6).

Remark 6.1 (DT+SOS folding Prover cost)**.** When it comes to DT+SOS, it is challenging to estimate the concrete costs of the folding Prover. We expect $\alpha \cdot \mathsf{L}$ to be the dominant cost, where recall L denotes the cost of computing the coefficients of $e(X)$ in [BC24], which has degree 9 in this case. Even using FFT's in the simplified scenario discussed in the footnote of Page 15 of [BC24], the cost would be over $\alpha \cdot 270m$ (Since computing $e(X)$ requires composing 9 homogeneous polynomials of degree $1, \ldots, 9$ with a linear polynomial, m times). To this, one must add the cost α times the cost $(3m, [M])$ (committing to a $3m$-sized vector with entries in $[M]$, where M is the largest entry in $\mathbf{t}$) plus the cost $\alpha \cdot \mathsf{P}_{\mathsf{sps}}$

of logUp-GKR's Prover, and other costs. Because of this, we expect the final cost to be higher than FLI+SOS.

Table 6: Approximate costs of Protostar+SOS, DT+SOS, and FLI+SOS when $m = 2^{17}, N = 2^{1024}, N^{1/c} = 2^8$, and $\alpha = 2c = 256$. For each scheme, we display the cost of the folding Prover (in field multiplications) and of the Prover for accumulated instances (in field multiplications and group operations, in columns 3 and 4). Regarding the latter, we describe Provers for instances accumulated with Protostar+SOS and DT+SOS below (Section 6). In the second and last column, we assume that a MSM-based PCS is used over the Pallas curve, and use Table 1 in [Hab22] to estimate the cost of these MSMs.
*The costs of DT+SOS's folding Prover are discussed in Remark 6.1
**These opening costs can be lowered, see the end of Section 6.

Scheme	**Folding Prover**	**Acc. Prover**	**Openings**	**Rounds**
Protostar+SOS	$\sim \alpha \cdot 2^{9.5} \cdot m$	$2^{12.39} \cdot m$	$\sim 2^{8.5} \cdot m$	$\log(m) + \alpha$
DT +SOS	*	$2^{12.25} \cdot m$	$\sim 2^{8.5} \cdot m$	$\log(m) + \alpha \cdot u \cdot \log(N^{1/c})$
FLI +SOS	$\begin{cases} \text{avg.:} & \sim c \cdot 2^{8.6} \cdot \rho \\ \text{worse:} & \sim c \cdot 2^{8.5} \cdot m \end{cases}$ $\rho := \min\{mn_f/N^{1/c}, m\}$	$2^{12.7} \cdot m$	$\sim 2^{16} \cdot m$**	$\log(m) + c$

Table 7: Approximate costs of Protostar+SOS, DT+SOS, and FLI+SOS when $m = 2^{17}, N = 2^{1024}, N^{1/c} = 2^4$, and $\alpha = 2c = 512$. We use the same assumptions as in the previous table.
*The costs of DT+SOS's folding Prover are discussed in Remark 6.1
**These opening costs can be lowered, see the end of Section 6.

Scheme	**Folding Prover**	**Acc. Prover**	**Openings**	**Rounds**
Protostar+SOS	$\sim \alpha \cdot 2^{9.5} \cdot m$	$2^{13.39} \cdot m$	$\sim 2^{8.5} \cdot m$	$\log(m) + \alpha$
DT +SOS	*	$2^{13.25} \cdot m$	$\sim 2^{8.5} \cdot m$	$\log(m) + \alpha \cdot u \cdot \log(N^{1/c})$
FLI +SOS	$\begin{cases} \text{avg.:} & \sim c \cdot 2^{8.6} \cdot \rho \\ \text{worse:} & \sim c \cdot 2^{8.5} \cdot m \end{cases}$ $\rho := \min\{mn_f/N^{1/c}, m\}$	$2^{13.59} \cdot m$	$\sim 2^{12} \cdot m$**	$\log(m) + c$

Note that if we were to use the Mova variation of FLI+SOS (see Section 5.5), the

concrete cost of the folding Prover would be dominated by $15cmN^{1/c} \sim c \cdot 2^{7.9} \cdot m$ field multiplications. This is shaving a $2^{0.6} \sim 1.52$ factor off the worst case. The protocol would have $\log(m) + 3c$ rounds instead of $\log(m) + c$. As we mentioned, this is useful if we know that we are using the folding step a large number of times.

Provers for instances that were accumulated with Protostar+SOS or DT+SOS. The protocols that prove accumulated instances for Protostar and DT are unspecified in the original papers. Recall that in Section 1.2 we described variations of Protostar and DT that first perform the SOS decomposition step of Lasso, which reduces the initial claim into α lookup instances, and then folding each of the α instances with α accumulated instances, using Protostar or DT, respectively. Accordingly, one can imagine a scheme for Protostar or DT that proves each of the α accumulated lookup instances separately. In Protostar, the accumulated claim is roughly a statement that the identities used in logUp hold (cf. [BC23] or Lemma 5 in [Hab22]). In DT, the same occurs, but this time the GKR variant of logUp is used [PH23]. For simplicity and for the sake of comparison, we assume that these claims can be proved, respectively, with logUp [Hab22] and logUp-GKR [PH23]. We emphasize that, to our knowledge, no actual protocols for proving accumulated instances have been formally described, and that they may be more complex than the schemes we just sketched (typically error terms would appear in the logUp equations). With this in mind, in Table 8 we lower bound Protostar's and DT's costs for proving accumulated instances as the cost of running logUp (when using Protostar) or logUp-GKR (when using DT) on each of the resulting α accumulated claims that arise when applying the decomposition step of Lasso. We set the cost of logUp as $21 \max\{N^{1/c}, m\}$ field multiplications, and of logUp-GKR as $19 \max\{N^{1/c}, m\}$ field multiplications. We derive these costs for logUp and logUp-GKR using [DT24] in Appendix D.

Remark 6.2. In their simplest forms, logUp [Hab22] and logUp-GKR [PH23] are lookups designed to handle instances where both the "small" and the "large" table have the same size. Both [Hab22, PH23] have more elaborate versions that handle $m > N^{1/c}$ by splitting m into $m/N^{1/c}$ "columns" (following the terminology in [Hab22, PH23]). When using this approach, as per [PH23], the Prover costs of Protostar and DT in Table 8 would incur a multiplicative overhead of $\approx \log(m/N^{1/c})$. Such a cost would make Protostar's and DT's costs for proving accumulated instances really high, and FLI would clearly be the best protocol. Because of this, and since it seems plausible, in our discussion we assume that there is a method that allows to treat the case $m > N^{1/c}$ by using one single column. We assume these improved protocols incur no overhead with respect to the original ones. The cost reflected in Table 8 corresponds to the dominant costs of such schemes. Further, the resulting cost for both Protostar and DT assume that we perform the sumchecks that prove logUp/logUp-GKR in parallel. In practice, this would lead to way too many evaluations, so the sumchecks would need to be batched. This results in an increase of concrete costs,

though not significant.

Table 8: Dominant costs of the protocols for proving accumulated instances with Protostar+SOS, DT+SOS, and FLI+SOS. We follow the same notation as in Tables 1 to 3. The opening costs refer to computing opening proofs of multilinear polynomials, possibly dense. We assume polynomial evaluation claims are batched together using standard techniques, hence, for each number of variables, there is only one opening proof to be generated. For both Protostar and Deep Thought, the costs are approximative and based on a simplified scheme, cf. Remark 6.2

Scheme	**Prover field mult**	**Openings**
Protostar +SOS	$21\alpha \max\{m, N^{1/c}\}$	$1 \cdot \log(m)$-variate, $1 \cdot \log(N^{1/c})$-variate
Deep Thought + SOS	$19\alpha \max\{m, N^{1/c}\}$	$1 \cdot \log(m)$-variate, $1 \cdot \log(N^{1/c})$-variate
FLI + SOS	$2m(N^{1/c}+1) + 3\alpha m \cdot n_f$	$1 \cdot \log(mN^{1/c})$-variate

Prover for accumulated instances' openings costs. FLI's scheme for proving accumulated instances requires proving an evaluation of a $\log(mN^{1/c})$-variate (sparse) multilinear polynomial. We assume that a curve-based commitment scheme is used, and that the opening proof bottleneck occurs when computing Multi-Scalar-Multiplication (MSM) of size $mN^{1/c}$ [10]. We remark that due to #2 in [Tha24], certain variations of Zeromorph [KT23] or HyperKZG [Set24] might make this step not be a bottleneck anymore. The price to pay is an increase of the folding Verifier work by $O(N^{1/c})$ (which can be configured to be small). Indeed, this is the approach planned by the Jolt team [Tha24]. In other words, here we analyze the costs of a naive selection of commitment scheme, but we remark that there are plausible alternatives.

[10] Following the benchmarks in Table 1 of [Hab22], we set the cost of this MSM when $mN^{1/c} > 2^{18}$ to be around $2^8 mN^{1/c}$ field multiplications, on the Pallas curve.

7 References

[AST24] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks fo virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 3–33, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-58751-1_1.

[AZ24] Arasu Arun and Michael Zhu. Jolt: Snarks for virtual machines via lookups. ZK Proof Standards, 2024. URL: https://www.youtube.com/live/RySXjCsLgXk.

[bar22] barryWhiteHat. Lookup singularity. ZKResearch, 2022. URL: https://zkresear.ch/t/lookup-singularity/65.

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In *Advances in Cryptology – ASIACRYPT 2023: 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4–8, 2023, Proceedings, Part II*, page 77–110, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-981-99-8724-5_3.

[BC24] Benedikt Bünz and Jessica Chen. Proofs for deep thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Paper 2024/325, 2024. URL: https://eprint.iacr.org/2024/325.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 681–710, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-84242-0_24.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Paper 2020/499, 2020. URL: https://eprint.iacr.org/2020/499.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1:3–40, 1991. doi:10.1007/BF01200056.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 677–706, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45721-1_24.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019. URL: https://eprint.iacr.org/2019/1021.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam Smith, editors, *Theory of Cryptography*, pages 31–60, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. doi:10.1007/978-3-662-53644-5_2.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 499–530, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-30617-4_17.

[CGG+23] Matteo Campanelli, Nicolas Gailly, Rosario Gennaro, Philipp Jovanovic, Mara Mihali, and Justin Thaler. Testudo: Linear time prover snarks with constant size proofs and square root size universal setup. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology – LATINCRYPT 2023*, pages 331–351, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-44469-2_17.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 738–768, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45721-1_26.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, pages 310–331. Tsinghua University Press, 2010.

[DGMV24] Nikolaos Dimitriou, Albert Garreta, Ignacio Manzur, and Ilia Vlasov. Mova: Nova folding without committing to error terms. Cryptology ePrint Archive, Paper 2024/1220, 2024. URL: https://eprint.iacr.org/2024/1220.

[DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023. URL: https://eprint.iacr.org/2023/1784.

[DP24] Benjamin E. Diamond and Jim Posen. Proximity testing with logarithmic randomness. *IACR Communications in Cryptology*, 1(1), 2024. doi:10.62056/aksdkp10.

[DT24] Quang Dao and Justin Thaler. More optimizations to sum-check proving. Cryptology ePrint Archive, Paper 2024/1210, 2024. URL: https://eprint.iacr.org/2024/1210.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763, 2022. URL: https://eprint.iacr.org/2022/1763.

[EG23] Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106, 2023. URL: https://eprint.iacr.org/2023/1106.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. doi:10.1007/3-540-47721-7_12.

[Hab22] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530, 2022. URL: https://eprint.iacr.org/2022/1530.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 723–732, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/129712.129782.

[KP23] Abhiram Kothapalli and Bryan Parno. Algebraic reductions of knowledge. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 669–701, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-38551-3_21.

[KS24] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 345–379, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-68403-6_11.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 359–388, Cham, 2022. Springer Nature Switzerland. doi:10.1007/978-3-031-15985-5_13.

[KT23] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023. URL: https://eprint.iacr.org/2023/917.

[Mic94] S. Micali. Cs proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 436–453, 1994. doi:10.1109/SFCS.1994.365746.

[PH23] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using gkr. Cryptology ePrint Archive, Paper 2023/1284, 2023. URL: https://eprint.iacr.org/2023/1284.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *Theory of Cryptography*, pages 222–242, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-36594-2_13.

[Set24] Srinath Setty. Hyperkzg, 2024. URL: https://github.com/microsoft/Nova/blob/main/src/provider/hyperkzg.rs.

[STW24] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 180–209, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-58751-1_7.

[Tha22] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge*. 01 2022. doi:10.1561/9781638281252.

[Tha24] Justin Thaler. Faq on jolt's initial implementation, 2024. URL: https://a16zcrypto.com/posts/article/faqs-on-jolts-initial-implementation/.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *Theory of Cryptography*, pages 1–18, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-78524-8_1.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3121–3134, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3560646.

[ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Paper 2022/1565, 2022. URL: https://eprint.iacr.org/2022/1565.

A Extended preliminaries

A.1 Interactive Proofs and Interactive Oracle Proofs

We recall the concepts of Interactive Proofs and Interactive Oracle Proofs [BSCS16]. At a high level, the former model is more suitable to describe our folding schemes. The latter model will be very useful for us to describe intermediate protocols that make heavy use of sumchecks (see Appendix A.2), which can be described as (polynomial) IOPs.

Following standard conventions, if $\mathbf{R}$ is an indexed relation, we call $(\mathbb{x}; \mathbb{w})$ such that $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}$ an instance-witness pair. We mean that $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness for $\mathbb{x}$. The set of valid witnesses for an instance $\mathbb{x}$ is denoted by $\mathbf{R}(\mathbb{i}, \mathbb{x}) := \{\mathbb{w} \mid (\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}\}$.

Definition A.1 (Interactive Proofs (IPs))**.** *A $2\mu + 1$-move, public coin IP $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ for an indexed relation $\mathbf{R}$ is an interactive protocol between two PPT algorithms P, V. The algorithm $\mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda)$ generates global parameters for the protocol Π, and the deterministic indexer outputs Verifier and Prover parameters given an index $(\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i})$. Both P and V take as input an instance $\mathbb{x}$; P additionally takes as a private input a witness $\mathbb{w} \in \mathbf{R}(\mathbb{i}, \mathbb{x})$. At round $i \in [\mu]$, P sends a message m_i, and V replies with a random challenge ρ_i. All of the random choices made by V are made public. At the last round, P sends a final message $m_{\mu+1}$ and V either accepts (returns 1) or rejects (returns 0). The output of V is denoted by the random variable $\langle \mathsf{P}(\mathsf{pp}, \mathbb{w}), \mathsf{V}(\mathsf{vp}) \rangle (\mathbb{x})$.*

The two standard security definitions that we use for the IPs in this paper are perfect completeness and knowledge soundness. We recall their definition next.

Definition A.2 (Perfect completeness of an IP)**.** *An IP $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ for the indexed relation $\mathbf{R}$ is said to be perfectly complete if for all $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}$:*

$$\Pr\left[\langle \mathsf{P}(\mathsf{pp}, \mathbb{w}), \mathsf{V}(\mathsf{vp}) \rangle (\mathbb{x}) = 1 \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array}\right] = 1$$

Definition A.3 (Knowledge soundness of an IP)**.** *An IP $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ for the indexed relation $\mathbf{R}$ is said to be knowledge sound with knowledge error $\epsilon : \mathbb{N} \to [0, 1]$ if there exists a PPT algorithm Ext (it runs in time polynomial in $|\mathbb{i}| + |\mathbb{x}|$) such that given oracle access to any pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, the following holds:*

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_2(\mathbb{i}, \mathsf{st}), \mathsf{V}(\mathsf{vp}) \rangle (\mathbb{x}) = 1) \\ \wedge \\ (\mathbb{i}, \mathbb{x}; \mathbb{w}) \notin \mathbf{R} \end{array} \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \\ \mathbb{w} \leftarrow \mathsf{Ext}^{\mathcal{A}_1, \mathcal{A}_2}(\mathsf{gp}, \mathbb{i}, \mathbb{x}) \end{array}\right] \leq \epsilon(|\mathbb{i}| + |\mathbb{x}|)$$

An IP is said to be knowledge sound, or an argument of knowledge, if ϵ is negligible in λ. If the adversaries are unbounded, then the IP is said to be a proof of knowledge.

We will use the notion of an accepting transcript for an IP when dealing with the forking lemma for folding schemes (see Section 3.5). We recall their definition next:

Definition A.4 (Accepting transcripts)**.** *The set of $2\mu + 1$ messages $\tau = (m_1, \rho_1, \ldots, m_{\mu+1})$, that is, $\mu + 1$ Prover messages $(m_1, \ldots, m_{\mu+1})$ and μ Verifier messages $(\rho_1, \ldots, \rho_\mu)$ resulting from an interaction of $\mathsf{P}(\mathsf{pp}, \mathbb{w}), \mathsf{V}(\mathsf{vp})$ in the IP is called a transcript.*

Such a transcript τ is said to be accepting if $\langle \mathsf{P}(\mathsf{pp}, \mathbb{w}), \mathsf{V}(\mathsf{vp}) \rangle(\mathbb{x}) = 1$.

Interactive Oracle Proofs (IOPs) [BSCS16] are information-theoretic proof systems that combine aspects of Interactive Proofs and Probabilistically Checkable Proofs, and also generalize the notion of Interactive PCPs.

Definition A.5 (Interactive Oracle Proofs (IOPs))**.** *A k-round public-coin IOP $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ is an interactive protocol between two PPT algorithms P, V. The algorithm $\mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda)$ generates global parameters for the protocol Π, and the deterministic indexer outputs Verifier and Prover parameters given an index $(\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i})$. Both P and V take as input an instance $\mathbb{x}$; P additionally takes as a private input a witness $\mathbb{w} \in \mathbf{R}(\mathbb{i}, \mathbb{x})$. At round $i \in [k-1]$, P sends oracle access to a proof string π_i and V replies with some public random element ρ_i. At the last round, P sends oracle access to one last proof string π_k. V then makes some queries to the proof strings $\pi_1, \ldots, \pi_k$ and decides whether to accept (returns 1) or reject (returns 0).*

Again, the security notions that we use in this paper are perfect completeness and knowledge soundness.

Definition A.6 (Perfect completeness of an IOP)**.** *An IOP $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ for an indexed relation $\mathbf{R}$ is said to be perfectly complete if for all $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}$:*

$$\Pr\left[\mathsf{V}^{\pi_1,\ldots,\pi_k}(\mathsf{vp}, \mathbb{x}, \rho_1, \ldots, \rho_{k-1}) = 1 \;\middle|\; \begin{array}{c} \pi_1 \leftarrow \mathsf{P}(\mathsf{pp}, \mathbb{x}; \mathbb{w}) \\ \vdots \\ \pi_k \leftarrow \mathsf{P}(\mathsf{pp}, \mathbb{x}; \mathbb{w}, \rho_1, \ldots, \rho_{k-1}) \end{array}\right] = 1$$

Definition A.7 (Knowledge soundness of an IOP)**.** *An IOP $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ for the indexed relation $\mathbf{R}$ is said to be knowledge sound with knowledge error $\epsilon : \mathbb{N} \to [0, 1]$ if there exists a PPT algorithm Ext (it runs in expected polynomial time in $|\mathbb{i}| + |\mathbb{x}|$) such that given*

oracle access to any pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$*, the following holds:*

$$\Pr\left[\begin{array}{c|c} \mathsf{V}^{\pi_1,\ldots,\pi_k}(\mathsf{vp}, \mathbb{x}, \rho_1, \ldots, \rho_{k-1}) = 1 & \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge & (\mathbb{i}, \mathbb{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathbb{i}, \mathbb{x}; \mathbb{w}) \notin \mathbf{R} & (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \\ & \pi_1 \leftarrow \mathcal{A}_2(\mathbb{x}) \\ & \vdots \\ & \pi_k \leftarrow \mathcal{A}_2(\mathbb{x}, \rho_1, \ldots, \rho_{k-1}) \\ & \mathbb{w} \leftarrow \mathsf{Ext}^{\mathcal{A}_1,\mathcal{A}_2}(\mathsf{gp}, \mathbb{i}, \mathbb{x}) \end{array}\right] \leq \epsilon(|\mathbb{i}| + |\mathbb{x}|)$$

An IOP is said to be knowledge sound if ϵ *is negligible in* λ*.*

A convenient idealized model of an IOP is given by Polynomial IOPs (PIOPs). These are public-coin IOPs for a particular type of oracle relations: the indices $\mathbb{i}$ and instances $\mathbb{x}$ can contain oracles to μ-variate polynomials over some field $\mathbb{F}$ with prescribed degree in each variable. The oracles specify μ and the degree in each variable. These oracles can be queried at arbitrary points in $\mathbb{F}^\mu$, the oracle returns the evaluation of the relevant polynomial at that point. The actual polynomials corresponding to the oracles in $\mathbb{i}$ (resp. $\mathbb{x}$) are contained in pp (resp. $\mathbb{w}$). The oracle to a polynomial f will be denoted by $[[f]]$. At each round, the Prover sends oracles to μ-variate polynomials over $\mathbb{F}$ with a prescribed degree in each variable, the Verifier still sends public randomness.

The notions of perfect completeness and knowledge soundness for PIOPs are the same, mutatis mutandis, as the corresponding notions for IOPs. This model is particularly useful because protocols are usually easily analyzed in the context where the Prover's messages consist of oracles to polynomials of prescribed degree.

We have focused on knowledge soundness as it is a stronger soundness notion, known to imply standard soundness. The contrapositive is not true in general. However, in the PIOP model the two notions are equivalent:

Lemma A.1 (Sound PIOPs are knowledge sound, Lemma 2.3 in [CBBZ23])**.** *Consider a* ϵ*-sound PIOP (meaning the PIOP has standard soundness error* $\epsilon(|\mathbb{i}| + |\mathbb{x}|)$*) for an oracle indexed relation* $\mathbf{R}$ *such that for all* $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathbf{R}$*,* $\mathbb{w}$ *consists only of polynomials such that the instance contains oracles to these polynomials. Then the PIOP has* ϵ *knowledge soundness, and the extractor runs in time* $O(|\mathbb{w}|)$*.*

Remark A.2 (Knowledge extractors for PIOPs)**.** The knowledge soundness extractor for the PIOP simply queries the oracles to each witness polynomials (in μ variables, and of degree at most d in each variable) at $(d+1)^\mu$ distinct points, and this determines a unique μ-variate polynomial of degree at most d in each variable.

We end this section with a short informal discussion on compilations of IPs, IOPs and PIOPs to other interactive and non-interactive arguments. A simple compilation from a PIOP to an interactive argument of knowledge replaces all oracles with polynomial commitments. Every query to an oracle is replaced with an invocation of the evaluation protocol of the polynomial commitment scheme at that query point. It is now well-known that if the polynomial commitment satisfies some extractability property, then this compilation turns a PIOP into a secure interactive argument of knowledge:

Lemma A.3 (PIOP compilation [BFS20, CHM+20])**.** *If the polynomial commitment scheme is extractable (see Definition 3.2)*[11]*, and the PIOP is knowledge sound, then the interactive protocol resulting from this compilation is a sound argument of knowledge.*

Remark A.4. Note that the IP obtained through this compilation is an IP for a slightly different relation than the PIOP oracle relation. It is the same relation, except that oracles to polynomials are replaced with commitments. The relation enforces additionally that the commitments to the polynomials in the Prover parameters/witness (supposedly those that correspond to the oracles in the oracle relation) are indeed equal to the commitments in the index/instance.

Compilations of IPs and IOPs to non-interactive arguments are related. The Fiat-Shamir transform [FS86], where the Prover derives randomness for the challenge at round i by evaluating a random oracle on the partial proof transcript $(\mathbb{x}, m_1, \rho_1, \ldots, m_i)$, transforms an IP into a non-interactive argument of knowledge (NIROA) in the Random Oracle Model (ROM). Usually, one then heuristically instantiates the random oracle with a cryptographic hash function. The way to transform an IOP into a NIROA is to replace the oracles with commitments (this transforms the IOP into an IP), and then using the Fiat-Shamir transform. This is called the BCS transformation (in [BSCS16], Merkle tree commitments are used).

In this paper, we focus on the interactive variants of the protocols we describe. By using the various compilations and heuristics, one can obtain non-interactive versions in practice.

A.2 The sumcheck protocol

For $n \geq 1$ and $d \geq 0$, we let $\mathbb{F}^{\leq d}[\mathbf{X}]$ be the set of multivariate polynomials in the variables $\mathbf{X} = (X_1, \ldots, X_n)$ with degree in each variable bounded by d. Let $\mathbf{R}_{\mathsf{sumcheck}}$ denote the following indexed oracle relation, which we call the *sumcheck relation*:

$$\mathbf{R}_{\mathsf{sumcheck}} := \left\{ \begin{pmatrix} \mathbb{i} = (\mathbb{F}, n, d), \\ \mathbb{x} = (c, [[f]]); \\ \mathbb{w} = f(\mathbf{X}) \end{pmatrix} \;\middle|\; \begin{array}{l} c \in \mathbb{F}, f(\mathbf{X}) \in \mathbb{F}^{\leq d}[\mathbf{X}], \\ \displaystyle\sum_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x}) = c \end{array} \right\}$$

[11] Note that we have defined extractable commitments only for multilinear polynomials, but this result holds for multivariate polynomial commitments just as well.

In words, indices in $\mathbf{R}_{\mathsf{sumcheck}}$ are a field $\mathbb{F}$, a number of variables n, and a degree bound d. An instance $\mathbb{x}$ consists of an oracle $[[f]]$ to a polynomial $f(\mathbf{X})$ on n variables over the field $\mathbb{F}$ with degree in each variable bounded by d, and the sum c of its values over the boolean hypercube $\mathbb{B}^n$. The witness $\mathbb{w}$ consists of the polynomial $f(\mathbf{X})$ corresponding to the oracle. The *sumcheck protocol* is a now standard PIOP for the sumcheck relation (for details about the construction, we refer to [Tha22, CBBZ23]). At a high level, the protocol proceeds in rounds as follows:

1. The Prover sends an oracle $[[f_1]]$ to the univariate polynomial $f_1(X) := \sum_{\mathbf{x}\in\mathbb{B}^{n-1}} f(X, \mathbf{x})$ The Verifier checks that $c = f_1(0) + f_1(1)$.
2. For $i = 1$ to $i = n - 1$,
   (a) The Verifier sends uniformly random $r_i \in \mathbb{F}$.
   (b) The Prover sends an oracle $[[f_{i+1}]]$ to the univariate polynomial $f_{i+1}(X) := \sum_{\mathbf{x}\in\mathbb{B}^{n-i-1}} f(r_1, \ldots, r_i, X, \mathbf{x})$.
   (c) The Verifier checks that $f_i(r_i) = f_{i+1}(0) + f_{i+1}(1)$.
3. In the end, the Verifier makes a single query to $[[f]]$ at $(r_1, \ldots, r_n)$ and checks that it agrees with $f_n(r_n)$.

The sumcheck protocol is perfectly complete, and has the following knowledge soundness error:

Lemma A.5. *The sumcheck protocol is a perfectly complete PIOP for* $\mathbf{R}_{\mathsf{sumcheck}}$ *with knowledge soundness error* $\epsilon = dn/|\mathbb{F}|$.

Now define the *zerocheck relation* as follows:

$$\mathbf{R}_{\mathsf{zerocheck}} := \left\{ \begin{pmatrix} \mathbb{i} = (\mathbb{F}, n, d), \\ \mathbb{x} = [[f]]; \\ \mathbb{w} = f(\mathbf{X}) \end{pmatrix} \;\middle|\; \begin{array}{l} c \in \mathbb{F}, f(\mathbf{X}) \in \mathbb{F}^{\leq d}[\mathbf{X}], \\ f(\mathbf{x}) = 0 \text{ for all } \mathbf{x} \in \mathbb{B}^n \end{array} \right\}.$$

This relation has the same indices as $\mathbf{R}_{\mathsf{sumcheck}}$, its instances are oracles to polynomials on n variables that vanish over the hypercube $\mathbb{B}^n$. The witness contains the polynomials corresponding to these oracles. One can use the sumcheck protocol to obtain a reduction of knowledge [KP23] from $\mathbf{R}_{\mathsf{sumcheck}}$ to $\mathbf{R}_{\mathsf{zerocheck}}$. This stems from the following observation:

Proposition A.6 (See [BFL91])**.** *Let* $n \geq 1$ *and let* $f(\mathbf{X}) \in \mathbb{F}[\mathbf{X}]$*, where* $\mathbf{X} = (X_1, \ldots, X_n)$*. Let* $\mathbf{r} \leftarrow \mathbb{F}^n$ *be uniformly sampled. Then, except with probability* $1/|\mathbb{F}|$*, we have that* $f(\mathbf{x}) = 0$ *for all* $\mathbf{x} \in \mathbb{B}^n$ *(i.e.* $((\mathbb{F}, n), [[f]]; f(\mathbf{X})) \in \mathbf{R}_{\mathsf{zerocheck}}$*) if and only if*

$$\sum_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{r}; \mathbf{x}) = 0,$$

i.e., if and only if $((\mathbb{F}, n), (0, [[f(\mathbf{X}) \cdot \widetilde{\mathsf{eq}}(\mathbf{r}; \mathbf{X})]]); f(\mathbf{X}) \cdot \widetilde{\mathsf{eq}}(\mathbf{r}; \mathbf{X})) \in \mathbf{R}_{\mathsf{sumcheck}}$.

Proof. Consider the multilinear polynomial $\widetilde{f}(\mathbf{X}) := \sum_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \mathbf{X})$. By uniqueness of MLEs, we have that $f(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathbb{B}^n$ if and only if $\widetilde{f}(\mathbf{X})$ equals the zero polynomial. Further, if $\widetilde{f}(\mathbf{X})$ is the zero polynomial, then $\widetilde{f}(\mathbf{r}) = 0$ for $\mathbf{r} \in \mathbb{F}^n$. On the other hand, if $\widetilde{f}(\mathbf{X})$ is note the zero polynomial, then by Schwartz-Zippel lemma, $\widetilde{f}(\mathbf{r}) \neq 0$ for all but at most $|\mathbb{F}|^{n-1}$ elements $\mathbf{r} \in \mathbb{F}^n$, and the result follows. □

Remark A.7. Note that the oracle for $f(\mathbf{X}) \cdot \widetilde{\mathsf{eq}}(\mathbf{r}; \mathbf{X})$ can also consist simply of $[[f]]$, because both the Prover and the Verifier can evaluate $\widetilde{\mathsf{eq}}(\mathbf{r}; \mathbf{X})$ at any field point efficiently. This is what happens in practice, this does not change the knowledge soundness of the protocol but affects the Prover and Verifier work.

We end this section by describing a small variation on the sumcheck protocol that we will use repeatedly, and proving its knowledge soundness in the PIOP model. This variation appears when the sum of a $m+n$-variate polynomial over the boolean hypercube $\mathbb{B}^n$ (contained in $\mathbb{B}^{m+n}$) is identically equal to a constant, that is:

$$\sum_{\mathbf{y}\in\mathbb{B}^n} f(\mathbf{X}, \mathbf{y}) \equiv c$$

The associated relation is as follows:

$$\mathbf{R}_{\mathsf{idsumcheck}} := \left\{ \begin{pmatrix} \mathbb{i} = (\mathbb{F}, m, n, d), \\ \mathbb{x} = (c, [[f]]); \\ \mathbb{w} = f(\mathbf{X}, \mathbf{Y}) \end{pmatrix} \middle| \begin{array}{c} c \in \mathbb{F}, f(\mathbf{X}, \mathbf{Y}) \in \mathbb{F}^{\leq d}[\mathbf{X}, \mathbf{Y}], \\ \mathbf{X} = (X_1, \ldots, X_m),\ \mathbf{Y} = (Y_1, \ldots, Y_n) \\ \sum_{\mathbf{y}\in\mathbb{B}^n} f(\mathbf{X}, \mathbf{y}) = c \end{array} \right\}$$

A simple PIOP for this relation can be described as follows:

1. The Verifier sends a random uniform $\mathbf{r} \in \mathbb{F}^m$.
2. The Prover and Verifier engage in the sumcheck protocol with instance $(c, [[f]])$ and witness $f'(\mathbf{Y}) = f(\mathbf{r}, \mathbf{Y})$. Queries to f' at $\mathbf{y}$ can be proved with oracle queries to $[[f]]$ at $(\mathbf{r}, \mathbf{y})$.
3. The Verifier accepts if and only if the sumcheck Verifier accepts.

Lemma A.8. *The protocol we just described is a perfectly complete PIOP for* $\mathbf{R}_{\mathsf{idsumcheck}}$. *It has soundness error* $\epsilon := d(m+n)/|\mathbb{F}|$, *and therefore it also has knowledge soundness error* ϵ.

Proof. Perfect completeness follows from the perfect completeness of the sumcheck protocol. Suppose $\sum_{\mathbf{y}\in\mathbb{B}^n} f(\mathbf{X}, \mathbf{y}) \neq c$. Then except with probability $dm/|\mathbb{F}|$ over the choice of $\mathbf{r}$, we have that $\sum_{\mathbf{y}\in\mathbb{B}^n} f(\mathbf{r}, \mathbf{y}) \neq c$. The soundness of the sumcheck protocol then implies that the sumcheck Verifier accepts with probability at most $dn/|\mathbb{F}|$. Knowledge soundness follows from Lemma A.1. □

There is also a batched version of sumcheck. If we have sumcheck instances $\sum_{\mathbf{x}\in\mathbb{B}^n} f_1(\mathbf{x}) = c_1, \ldots, \sum_{\mathbf{x}\in\mathbb{B}^n} f_\nu(\mathbf{x}) = c_\nu$, we can verify them all at once by performing a sumcheck of the form:

$$\sum_{\mathbf{x}\in\mathbb{B}^n} \sum_{i\in[\nu]} \gamma^{i-1} \cdot f_i(\mathbf{x}) = c_1 + \cdots + \gamma^{\nu-1} \cdot c_\nu$$

for a randomly chosen $\gamma \in \mathbb{F}$. The resulting protocol is also knowledge sound (this can be expressed in the language of reductions of knowledge [KP23]). This is a standard technique, and we omit the proof of knowledge soundness.

Remark A.9. We have mentioned that the sumcheck protocol can be considered as a PIOP for the sumcheck relation, and this makes the analysis of its knowledge soundness particularly simple. In practice, the sumcheck protocol is implemented with the Prover sending (a representation of) the actual polynomial corresponding to the oracle at each intermediate round. This does not change the analysis of the knowledge soundness of the protocol, but does affect the field operation work (and the communication complexity) of the Prover and Verifier. When we examine the costs of the sumcheck protocol, and the strategies for reducing them, we mean the version of sumcheck where the Prover sends actual polynomials at each intermediate round. In this paper, when we write the costs of protocols using the sumcheck as a subroutine, we will consider the sumcheck IP that is obtained from the sumcheck where the Prover sends full polynomials at intermediate rounds, and where the oracle to the multivariate polynomial in the instance is replaced with a polynomial commitment (that is extractable).

B Comparing other regimes for m and N

The case where m is small and $N^{1/c}$ is large. Assuming $N^{1/c}$ is large and both m and n_f are small, then in general FLI's commitment cost is very small (on average), coming from committing to c vectors of, mostly, binary elements. In this scenario, we can reasonably assume that FLI's folding cost is dominated by $m \deg(g)(\alpha + |g|)$ field operations. As shown in Tables 2 and 3, in this parameter regime, FLI results in the most efficient folding scheme available, for many parameter choices. Further, towards building an IVC/PCD scheme, FLI's Verifier is the cheapest among the schemes in Tables 2 and 3.

If n_f is large, then ρ may degenerate to m, in which case FLI's folding Prover must commit to c size-m vectors of arbitrarily sized field elements. Still, Protostar must commit to $2\alpha = 2 \cdot k \cdot c$ size-m such vectors This is roughly $2k$ times more expensive than FLI's commitment costs. Further, we note that the overall number of commitments to large elements over the n'_f folding steps is, in FLI, on average:

$$c\frac{m}{N^{1/c}}(1 + 2 + \ldots + n_f) \approx \frac{cn_f^2 m}{2N^{1/c}},$$

while in Protostar it is $2\alpha n_f m$.

The case when both m and $N^{1/c}$ are large. So far, we have only discussed the case when $mN^{1/c}$ is roughly our computation target. This limited our analysis to settings where m and/or $N^{1/c}$ were "small". The reason for this analysis is that FLI's protocol for proving accumulated instances has cost $O(mN^{1/c})$, and thus cannot handle scenarios where both m and $N^{1/c}$ are large. On the other hand, Protostar and Deep Thought do not present the quadratic term $mN^{1/c}$, and so it is only fair to remark that one may use them in this regime, while FLI is not usable in this case.

We note however that, when dealing with SOS-decomposable tables as the ones from Jolt, it is always possible to select c so that $N^{1/c}$ is small. Hence, when looking into such tables, there is always the option of selecting parameters as in Section 6 and use FLI.

C Deferred proofs

C.1 Proof of Lemma 4.2

Proof of Lemma 4.2. Perfect completeness follows from the perfect completeness of the sumcheck protocol.

Now suppose there are adversaries $\mathcal{A}_1, \mathcal{A}_2$ such that $\mathcal{A}_1$ consumes global parameters and produces an index instance pair $(\mathtt{i}, \mathtt{x})$; $\mathcal{A}_2$ takes as input $(\mathtt{i}, \mathtt{x})$ and interacts with the Verifier according to the PIOP we described. Note that the index instance pair is of the form $((\mathbb{F}, N, m), [[M]])$, with $m \leq N < \mathrm{char}(\mathbb{F})$. Suppose additionally that the Verifier accepts.

This means in particular that each sumcheck Verifier accepted. By knowledge soundness of the sumcheck and of our variation on the sumcheck (Lemma A.8), there exists two efficient extractors Ext_1 (resp. Ext_2) that given access to the global parameters, $\mathcal{A}_1, \mathcal{A}_2$ and on input $\mathtt{i}, \mathtt{x}$ output $M_1 \in \mathbb{F}^{\leq 1}[\mathbf{X}, \mathbf{Y}]$ (resp. $M_2 \in \mathbb{F}^{\leq 1}[\mathbf{X}, \mathbf{Y}]$) such that

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} (M_1(\mathbf{x}, \mathbf{y})^2 - M_1(\mathbf{x}, \mathbf{y})) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}; \mathbf{x}, \mathbf{y}) = 0$$

(resp. $\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M_2(\mathbf{X}, \mathbf{y}) \equiv 1$), except with probability at most $3\log(mN)/|\mathbb{F}|$ (resp. $\log(mN)/|\mathbb{F}|$). But M_1 and M_2 are obtained by interpolating queries to the same oracle $[[M]]$ (see Remark A.2), so that $M_1 = M_2$, call their common value M[12]. We have that:

$$\sum_{\mathbf{x} \in \mathbb{B}^{\log(m)}} \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} (M(\mathbf{x}, \mathbf{y})^2 - M(\mathbf{x}, \mathbf{y})) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}; \mathbf{x}, \mathbf{y}) = 0$$

[12] This also means that we may run the extractor on any of the two sumcheck instances, for example we may only run Ext_2. Such an extracted witness will be a valid witness for the first sumcheck except with probability at most $3\log(mN)/|\mathbb{F}|$.

$$\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M(\mathbf{X},\mathbf{y}) \equiv 1$$

and the first equality implies Eq. (7) except with probability $1/|\mathbb{F}|$ over the choice of $\boldsymbol{\beta}$ by Proposition A.6. Therefore, we can extract a valid witness except with probability at most $(4\log(mN)+1)/|\mathbb{F}|$. □

C.2 Proof of Lemma 4.3

Proof of Lemma 4.3. We look at perfect completeness first. Let

$$(\mathbb{i},\mathbb{x};\mathbb{w}) = ((\mathbb{F},N,m),(\overline{M_{\mathsf{acc}}},\overline{E_{\mathsf{acc}}},\mu);(M_{\mathsf{acc}},E_{\mathsf{acc}})) \in \mathbf{R}^{\mathsf{acc}}_{\mathsf{elem}}$$

and

$$(\mathbb{i},\mathbb{x}';\mathbb{w}') = ((\mathbb{F},N,m),\overline{M};M) \in \mathbf{R}_{\mathsf{elem}}$$

be any tuples in each relation. Again, we suppose that both the Prover and Verifier have access to $(\mathbb{F},N,m),(\overline{M_{\mathsf{acc}}},\overline{E_{\mathsf{acc}}},\mu),\overline{M}$, and that the Prover has access to $M_{\mathsf{acc}},E_{\mathsf{acc}},M$. Suppose that at the end of the protocol honestly, an honest Prover and Verifier output $(\overline{M'_{\mathsf{acc}}},\overline{E'_{\mathsf{acc}}},\mu')$ and the honest Prover outputs $(M'_{\mathsf{acc}},E'_{\mathsf{acc}})$. We must prove that the following holds:

$$\begin{aligned}
\overline{E'_{\mathsf{acc}}} &= \mathsf{cm}(E'_{\mathsf{acc}})\\
\overline{M'_{\mathsf{acc}}} &= \mathsf{cm}(M'_{\mathsf{acc}})\\
\forall(\mathbf{x},\mathbf{y})\in\mathbb{B}^{\log(mN)},\ M'_{\mathsf{acc}}(\mathbf{x},\mathbf{y})^2 &= M'_{\mathsf{acc}}(\mathbf{x},\mathbf{y}) + E'_{\mathsf{acc}}(\mathbf{x},\mathbf{y})\\
\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M'_{\mathsf{acc}}(\mathbf{X},\mathbf{y}) &= (1+\mu')
\end{aligned}$$

The first two equations hold by construction and by the homomorphic property of the commitment scheme. For the next equalities, we have for all $(\mathbf{x},\mathbf{y})\in\mathbb{B}^{\log(mN)}$:

$$\begin{aligned}
M'_{\mathsf{acc}}(\mathbf{x},\mathbf{y})^2 &= (M_{\mathsf{acc}}(\mathbf{x},\mathbf{y}) + \alpha\cdot M(\mathbf{x},\mathbf{y}))^2\\
&= M_{\mathsf{acc}}(\mathbf{x},\mathbf{y})^2 + 2\alpha\cdot M_{\mathsf{acc}}(\mathbf{x},\mathbf{y})M(\mathbf{x},\mathbf{y}) + \alpha^2\cdot M(\mathbf{x},\mathbf{y})^2\\
&= M_{\mathsf{acc}}(\mathbf{x},\mathbf{y}) + E_{\mathsf{acc}}(\mathbf{x},\mathbf{y}) + 2\alpha\cdot M_{\mathsf{acc}}(\mathbf{x},\mathbf{y})M(\mathbf{x},\mathbf{y}) + \alpha^2\cdot M(\mathbf{x},\mathbf{y})\\
&= (M_{\mathsf{acc}}(\mathbf{x},\mathbf{y}) + \alpha\cdot M(\mathbf{x},\mathbf{y})) + E_{\mathsf{acc}}(\mathbf{x},\mathbf{y})\\
&+ \alpha\cdot(2M_{\mathsf{acc}}(\mathbf{x},\mathbf{y})M(\mathbf{x},\mathbf{y}) - M(\mathbf{x},\mathbf{y})) + \alpha^2\cdot M(\mathbf{x},\mathbf{y})\\
&= M'_{\mathsf{acc}}(\mathbf{x},\mathbf{y}) + E'_{\mathsf{acc}}(\mathbf{x},\mathbf{y})
\end{aligned}$$

Finally, we have that:

$$\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M'_{\mathsf{acc}}(\mathbf{X},\mathbf{y}) = \sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M_{\mathsf{acc}}(\mathbf{X},\mathbf{y}) + \alpha\cdot\sum_{\mathbf{y}\in\mathbb{B}^{\log(N)}} M(\mathbf{X},\mathbf{y})$$

$$\equiv 1 + \mu + \alpha = 1 + \mu'$$

We now move on to knowledge soundness. The argument is similar to the one in [KST22]. Let $(\overline{M_1}, \overline{E_1}, \mu_1)$ and $\overline{M_2}$ be two instances for $\mathbf{R}_{\mathsf{elem}}^{\mathsf{acc}}$ and $\mathbf{R}_{\mathsf{elem}}$, respectively. The idea is to use Lemma 3.1 to show that an extractor can output satisfying witnesses for these instances given a large enough tree of accepting transcripts for the folding protocol, and the corresponding folded instance-witness pairs. More precisely, we show that the extractor Ext can output a satisfying witness given three accepting transcripts (τ_1, τ_2, τ_3) with the same initial message from the Prover: the commitment to the cross term $\overline{T}$. Note that a transcript τ_i, for $i = 1, 2, 3$, additionally comes attached with a satisfying witness which we denote by $\tau_i.(M, E)$, and the Verifier's random element $\tau_i.\alpha$. Interpolating the points $(\tau_1.\alpha, \tau_1.M), (\tau_2.\alpha, \tau_2.M)$, Ext finds M_1, M_2 such that:

$$\text{For } i = 1, 2, \quad M_1 + \tau_i.\alpha \cdot M_2 = \tau_i.M$$

Similarly, interpolating $(\tau_1.\alpha, \tau_1.E), (\tau_2.\alpha, \tau_2.E), (\tau_3.\alpha, \tau_3.E)$, Ext finds E_1, E_2 and a cross term T such that:

$$E_1 + \tau_i.\alpha \cdot T + \tau_i.\alpha^2 \cdot E_2 = \tau_i.E, \quad i = 1, 2, 3.$$

We now show that the witness elements we just found are valid openings to the commitments in the instances. Because $\tau_i.M$ is part of a satisfying witness, by construction and the homomorphic property of the commitment, we find that for $i = 1, 2$:

$$\begin{aligned} \overline{M_1} + \tau_i.\alpha \cdot \overline{M_2} &= \mathsf{cm}(\tau_i.M) \\ &= \mathsf{cm}(M_1 + \tau_i.\alpha \cdot M_2) \\ &= \mathsf{cm}(M_1) + \tau_i.\alpha \cdot \mathsf{cm}(M_2) \end{aligned}$$

and this implies that $\mathsf{cm}(M_i) = \overline{M_i}$ for $i = 1, 2$. Similarly we must have for $i = 1, 2, 3$:

$$\begin{aligned} \overline{E_1} + \tau_i.\alpha \cdot \overline{T} + \tau_i.\alpha^2 \cdot \overline{M_2} &= \mathsf{cm}(\tau_i.E) \\ &= \mathsf{cm}(E_1 + \tau_i.\alpha \cdot T + \tau_i.\alpha^2 \cdot E_2) \\ &= \mathsf{cm}(E_1) + \tau_i.\alpha \cdot \mathsf{cm}(T) + \tau_i.\alpha^2 \cdot \mathsf{cm}(E_2) \end{aligned}$$

and this implies that $\mathsf{cm}(E_1) = \overline{E_1}$, $\mathsf{cm}(T) = \overline{T}$ and $\mathsf{cm}(E_2) = \overline{M_2}$. By the binding property of the commitment scheme, $E_2 = M_2$ except with probability $\mathrm{negl}(\lambda)$. We must now argue that (M_1, E_1, μ_1) and M_2 satisfy the remaining conditions in their respective relations. First, note the following chain of equalities:

$$\begin{aligned} \mathsf{cm}(M_1 + \tau_3.\alpha \cdot M_2) &= \mathsf{cm}(M_1) + \tau_3.\alpha \cdot \mathsf{cm}(M_2) \\ &= \overline{M_1} + \tau_3.\alpha \cdot \overline{M_2} \\ &= \mathsf{cm}(\tau_3.M) \end{aligned}$$

By the binding property of the commitment scheme, $M_1 + \tau_3.\alpha \cdot M_2 = \tau_3.M$ except with probability $\mathrm{negl}(\lambda)$. Because $(\tau_i.M, \tau_i.E)$ is a satisfying witness for $i = 1, 2, 3$, we have that:

$$\tau_i.M \circ \tau_i.M = \tau_i.M + \tau_i.E \tag{12}$$

$$\tau_i.M \cdot \mathbf{1}^T = (1 + \tau_i.\mu) \cdot \mathbf{1}^T. \tag{13}$$

From Eq. (12) we have, for $i = 1, 2, 3$,

$$\begin{aligned}&(M_1 + \tau_i.\alpha \cdot M_2) \circ (M_1 + \tau_i.\alpha \cdot M_2)\\ =&(M_1 + \tau_i.\alpha \cdot M_2) + (E_1 + \tau_i.\alpha \cdot T + \tau_i.\alpha^2 \cdot M_2)\end{aligned}$$

which upon expanding and interpolating, shows that $M_1 \circ M_1 = M_1 + E_1$ and $M_2 \circ M_2 = M_2$. Similarly interpolating the equations given by Eq. (13), we find that $M_1 \cdot \mathbf{1}^T = (1 + \mu_1) \cdot \mathbf{1}^T$ and $M_2 \cdot \mathbf{1}^T = \mathbf{1}^T$. □

C.3 Proof of Lemma 5.1

Proof of Lemma 5.1. We look at perfect completeness first. Let

$$(\mathbb{i}, \mathbb{x}; \mathbb{w}) = ((\mathbb{F}, N, m, \mathbf{t}), (\overline{\mathbf{a}_{\mathsf{acc}}}, \overline{M_{\mathsf{acc}}}, \overline{E_{\mathsf{acc}}}, \mu); (\mathbf{a}_{\mathsf{acc}}, M_{\mathsf{acc}}, E_{\mathsf{acc}})) \in \mathbf{R}_1^{acc}$$

and

$$(\mathbb{i}, \mathbb{x}'; \mathbb{w}') = ((\mathbb{F}, N, m, \mathbf{t}), (\overline{\mathbf{a}}, \overline{M}); (\mathbf{a}, M)) \in \mathbf{R}_{\mathsf{CmMAILook}}$$

be any tuples in each relation. Again, we suppose that both the Prover and Verifier have access to $(\mathbb{F}, N, m), \mathbf{t}, (\overline{\mathbf{a}_{\mathsf{acc}}}, \overline{M_{\mathsf{acc}}}, \overline{E_{\mathsf{acc}}}, \mu), (\overline{\mathbf{a}}, \overline{M})$, and that the Prover has access to $\mathbf{a}_{\mathsf{acc}}, M_{\mathsf{acc}}, E_{\mathsf{acc}}, \mathbf{a}, M$. Suppose that at the end of following the protocol honestly, the Prover and Verifier output $(\overline{\mathbf{a}'_{\mathsf{acc}}}, \overline{M'_{\mathsf{acc}}}, \overline{E'_{\mathsf{acc}}}, \mu')$ and the Prover outputs $(\mathbf{a}_{\mathsf{acc}}, M_{\mathsf{acc}}, E_{\mathsf{acc}})$. By perfect completeness of $\mathscr{P}_1$ (Lemma 4.3), we know that $((\mathbb{F}, N, m), (\overline{M'_{\mathsf{acc}}}, \overline{E'_{\mathsf{acc}}}, \mu); (M'_{\mathsf{acc}}, E'_{\mathsf{acc}})) \in \mathbf{R}_{\mathsf{elem}}^{\mathsf{acc}}$, so we need only prove that:

$$\begin{aligned}\overline{\mathbf{a}'_{\mathsf{acc}}} &= \mathsf{cm}(\mathbf{a}'_{\mathsf{acc}}),\\ \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M'_{\mathsf{acc}}(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) &= \mathbf{a}'_{\mathsf{acc}}(\mathbf{X}).\end{aligned}$$

The first property follows by construction and the homomorphic property of the commitment scheme. For the second property, we have by construction:

$$\begin{aligned}\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M'_{\mathsf{acc}}(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) &= \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} (M_{\mathsf{acc}}(\mathbf{X}, \mathbf{y}) + \alpha M(\mathbf{X}, \mathbf{y})) \cdot \mathbf{t}(\mathbf{y})\\ &= \mathbf{a}_{\mathsf{acc}}(\mathbf{X}) + \alpha \cdot \mathbf{a}(\mathbf{X})\end{aligned}$$

$$= \mathbf{a}'_{\mathsf{acc}}(\mathbf{X})$$

Knowledge soundness of $\mathscr{F}_1$ almost follows from knowledge soundness of $\mathscr{P}_1$. Let $(\overline{\mathbf{a}_1}, \overline{M_1}, \overline{E_1}, \mu)$ and $(\overline{\mathbf{a}_2}, \overline{M_2})$ be any instances for the relations $\mathbf{R}_1^{acc}$ and $\mathbf{R}_{\mathsf{CmMAILook}}$ respectively. We have seen in Lemma 4.3 that an extractor Ext can output satisfying witnesses for $\mathbf{R}_{\mathsf{elem}}$ and $\mathbf{R}_{\mathsf{elem}}^{acc}$ from three accepting transcripts (τ_1, τ_2, τ_3) with the same initial message from the Prover (the commitment to the cross-term). Here, transcripts additionally come attached with a satisfying witness $\tau_i.(\mathbf{a}, M, E)$. It remains to show that the extractor can output $\mathbf{a}_1$ and $\mathbf{a}_2$ such that $(\mathbf{a}_1, M_1, E_1)$ and $(\mathbf{a}_2, M_2)$ are satisfying witnesses for their respective relations. But from interpolating $(\tau_1.\alpha, \tau_1.\mathbf{a}), (\tau_2.\alpha, \tau_2.\mathbf{a})$, Ext finds $\mathbf{a}_1, \mathbf{a}_2$ such that for $i = 1, 2$:

$$\mathbf{a}_1 + \tau_i.\alpha \cdot \mathbf{a}_2 = \tau_i.\mathbf{a}$$

By construction and by the homomorphic property of the commitment, we have that $\mathsf{cm}(\mathbf{a}_i) = \overline{\mathbf{a}_i}$ for $i = 1, 2$. Finally, because $(\mathbf{a}, M, E)$ form a satisfying witness, we find that for $i = 1, 2$:

$$\begin{aligned}\mathbf{a}_1(\mathbf{X}) + \tau_i.\alpha \cdot \mathbf{a}_2(\mathbf{X}) = \tau_i.\mathbf{a}(\mathbf{X}) &= \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} \tau_i.M(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) \\ = \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M_1(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) &+ \tau_i.\alpha \cdot \sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M_2(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y})\end{aligned}$$

Interpolating, we obtain $\sum_{\mathbf{y} \in \mathbb{B}^{\log(N)}} M_i(\mathbf{X}, \mathbf{y}) \cdot \mathbf{t}(\mathbf{y}) = \mathbf{a}_i(\mathbf{X})$ for $i = 1, 2$. □

C.4 Proof of Lemma 5.4

Proof of Lemma 5.4. Perfect completeness follows from perfect completeness of the sumcheck protocol and the homomorphic property of the commitment scheme.

For knowledge soundness, we use Lemma 3.1 again. Let

$$(\overline{M^{\mathsf{acc}}}, \overline{M^{\mathsf{acc}}}_1, \ldots, \overline{M^{\mathsf{acc}}}_c, \overline{E_{\mathsf{acc}}}, \overline{\mathbf{a}_{\mathsf{acc}}}, \mathbf{r}, \mu, c_1, \ldots, c_\alpha) \text{ and } (\overline{M_1}, \ldots, \overline{M_c}, \overline{\mathbf{a}_2})$$

be two instances for $\mathbf{R}^{\mathsf{accSOS}}$ and $\mathbf{R}^{\mathsf{SOS}}$ respectively. Suppose we have three accepting transcripts τ_1, τ_2, τ_3 that are identical until the Prover message where it sends the commitment to the cross-terms (included). This means in particular that the sumcheck part of each transcript is identical, as well as the commitments to the cross terms. The transcripts additionally come attached with a satisfying witness for $\mathbf{R}^{\mathsf{accSOS}}$ which we denote by $\tau_i.(M^{\mathsf{acc}\prime}, M_1^{\mathsf{acc}\prime}, \ldots, M_c^{\mathsf{acc}\prime}, E'_{\mathsf{acc}}, \mathbf{a}'_{\mathsf{acc}})$, and the Verifier's random element $\tau_i.\eta_j$, for $j \in [c]$.

Note that in protocol $\mathscr{F}_1^{SOS}$, after the sumcheck, there are essentially two procedures going on in parallel for $i \in [c]$:

- M_i is folded into M^{acc} with protocol $\mathscr{P}_1$ (see Lemma 4.3).

- M_i^{acc} and M_i are linearly combined as $M_i^{\mathsf{acc}} + \tau_j.\eta_i \cdot M_i$.

We know how to extract witnesses from the second point by interpolating, this gives us witnesses M_i^{acc}, M_i that commit to the elements $\overline{M_i^{\mathsf{acc}}}, \overline{M_i}$ in the instance, respectively. By applying the knowledge soundness extractor of $\mathscr{P}_1$ repeatedly (from $i = c$ to $i = 1$), we are able to extract M^{acc} and E_{acc} that commit to the elements $\overline{M^{\mathsf{acc}}}, \overline{E_{\mathsf{acc}}}$ in the instances. Further M^{acc} and E_{acc} satisfy the conditions in $\mathbf{R}_{\mathsf{elem}}^{\mathsf{acc}}$. Similarly because the update procedure of the small table $\mathbf{a}_{\mathsf{acc}}$ is a simple linear combination, we can also extract the relevant witnesses.

Finally, by the soundness of the sumcheck protocol, we know that except with negligible probability in the security parameter, the witnesses we have found satisfy:

$$\sum_{\mathbf{x}\in\mathbb{B}^{\log(m)}} (\mathbf{a}(\mathbf{x}) - g(M_1(\mathbf{x})T_1, \ldots, M_c(\mathbf{x})T_\alpha)) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\beta}, \mathbf{x}) = 0$$

$$\forall i \in [c],\ M_i^{\mathsf{acc}}(\mathbf{r}) = c_i$$

$$\mathbf{a}_{\mathsf{acc}}(\mathbf{r}) = d$$

and e.w.n.p. (Proposition A.6) the first equality implies that for all $\mathbf{x} \in \mathbb{B}^{\log(m)}$, $\mathbf{a}(\mathbf{x}) - g(M_1(\mathbf{x})T_1, \ldots, M_c(\mathbf{x})T_\alpha) = 0$. □

D The cost of proving accumulated instances with Protostar+SOS and DT+SOS

As discussed in Section 6, we use an oversimplification and estimate the cost of proving an accumulated instance with Protostar+SOS (resp. DT+SOS) by estimating the cost of proving α sumchecks appearing in logUp (resp. logUp-GKR) when applied to a lookup of a table of size $\max\{m, N^{1/c}\}$ into a table of the same size. In turn, we estimate the cost of proving a single such sumcheck using the new optimized costs in [DT24].

The cost of logUp [Hab22]. For logUp, the sumcheck for a lookup of a small table $\mathbf{a}$ into a big table $\mathbf{t}$ (both of size $M := \max\{m, N^{1/c}\}$) is of the form:

$$\sum_{\mathbf{x}\in\mathbb{B}^{\log(M)}} h_1(\mathbf{x}) + h_2(\mathbf{x}) + \gamma \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \boldsymbol{\beta}) \cdot (h_1(\mathbf{x}) \cdot (\alpha + \mathbf{t}(\mathbf{x})) - \mathsf{m}(\mathbf{x}))) +$$
$$\gamma^2 \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \boldsymbol{\beta}) \cdot (h_2(\mathbf{x}) \cdot (\alpha + \mathbf{a}(\mathbf{x})) - 1)) = 0 \qquad (\star)$$

where $\alpha \in \mathbb{F}$ and $\boldsymbol{\beta} \in \mathbb{F}^{\log(M)}$ are chosen at random, m is the "multiplicity function" (c.f. [Hab22], this functions indicates how many times each entry $\mathbf{t}(\mathbf{x})$ is looked up) and:

$$\forall \mathbf{x} \in \mathbb{B}^{\log(M)},\ h_1(\mathbf{x}) := \mathsf{m}(\mathbf{x})/(\alpha + \mathbf{t}(\mathbf{x}))$$

$$h_2(\mathbf{x}) := -1/(\alpha + \mathbf{a}(\mathbf{x}))$$

To apply the sumcheck optimizations of [DT24], we need all the evaluations of h_1, h_2. Batch inversion techniques allow to compute the vector of inverses of $(\alpha+\mathbf{t}(\mathbf{x}), \alpha+\mathbf{a}(\mathbf{x}))_{\mathbf{x}}$ in about $6M$ multiplications, and 1 inversion. We can then use M multiplications to compute all evaluations of h_1, h_2 over the hypercube. Following [DT24], the Prover in sumcheck in $(\star)$ performs:

- $2M$ field multiplications for the terms $h_1(\mathbf{x})$ and $h_2(\mathbf{x})$.
- $6M+O(\sqrt{M})$ field multiplications for the term $\gamma \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \boldsymbol{\beta}) \cdot (h_1(\mathbf{x}) \cdot (\alpha+\mathbf{t}(\mathbf{x})) - \mathsf{m}(\mathbf{x})))$.
- $6M$ field multiplications for the term $\gamma^2 \cdot \widetilde{\mathsf{eq}}(\mathbf{x}; \boldsymbol{\beta}) \cdot (h_2(\mathbf{x}) \cdot (\alpha + \mathbf{a}(\mathbf{x})) - 1))$.

For a total cost of $21M + O(\sqrt{M})$ field multiplications.

The cost of logUp-GKR [PH23]. For logUp-GKR, for a lookup of a small table $\mathbf{a}$ into a big table $\mathbf{t}$ (both of size $M := \max\{m, N^{1/c}\}$), there are k sumchecks (for $1 \leq k \leq \log(M) - 1$) of the form:

$$\sum_{\mathbf{x} \in \mathbb{B}^k} \widetilde{\mathsf{eq}}(\mathbf{x}; \boldsymbol{\beta}) \cdot (p_{k+1}(\mathbf{x}, 1) \cdot q_{k+1}(\mathbf{x}, -1) + p_{k+1}(\mathbf{x}, -1) \cdot q_{k+1}(\mathbf{x}, 1) + \gamma_k \cdot q_{k+1}(\mathbf{x}, 1) \cdot q_{k+1}(\mathbf{x}, -1)) = p_k(\boldsymbol{\beta}) + \gamma_k \cdot q_k(\boldsymbol{\beta}) \qquad (\dagger)$$

where $\alpha \in \mathbb{F}$ and $\boldsymbol{\beta} \in \mathbb{F}^k$ are chosen at random, and:

$$p(\mathbf{X}, Y) := Y \cdot \mathsf{m}(\mathbf{X}) - (1 - Y)$$
$$q(\mathbf{X}, Y) := Y \cdot (\alpha - \mathbf{t}(\mathbf{X})) + (1 - Y) \cdot (\alpha - \mathbf{a}(\mathbf{X}))$$
$$\forall 1 \leq k \leq \log(M) - 1, \forall \mathbf{x} \in \mathbb{B}^k, \quad \frac{p_k(\mathbf{x})}{q_k(\mathbf{x})} = \sum_{\mathbf{y} \in \mathbb{B}^{\log(M)-k}} \frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{x}, \mathbf{y})}$$

At layer k, following [DT24] and assuming that we have all relevant values of p_{k+1}, q_{k+1}, the Prover can perform the sumcheck in $(\dagger)$ in $16 \cdot 2^k + O(2^{k/2})$ field multiplications. Summing over k, this leads to $16M + O(\sqrt{M})$ field multiplications overall. To compute all necessary values of the p_k, q_k for all k, we need $3M$ multiplications (this is because of the linear relationships between p_k, q_k and p_{k+1}, q_{k+1}, see [PH23]). All in all, we estimate that the Prover in logUp-GKR performs no more than $19M + O(\sqrt{M})$ field multiplications.