

Quasar: A Field-Agnostic Polynomial Commitment Scheme with Polylogarithmic Verification from Quasi-Abelian Codes

Yuhao Jia¹, Zhe Li², Chaoping Xing¹, Yizhou Yao¹, Chen Yuan¹

¹ School of Computer Science,
Shanghai Jiao Tong University, Shanghai, China
{jyh1529400,xingcp,yaoyizhou0620,chen_yuan}@sjtu.edu.cn

² Institute of Mathematics and Interdisciplinary Sciences,
Xidian University, Xi'an, China
lizh0048@e.ntu.edu.sg

Abstract. Polynomial commitment schemes (PCSs) are fundamental building blocks of modern zkSNARKs and often dominate their concrete prover and verifier costs. We introduce **Quasar**, a field-agnostic PCS for multilinear polynomials that combines Quasi-Abelian (QA) codes with BaseFold (Zeilberger et al., CRYPTO 2024) through code switching (Ron-Zewi and Rothblum, JACM 2024). For a polynomial of length N and security parameter λ , **Quasar** achieves concretely fast $O(N \log N)$ commitment, $O(N)$ evaluation time, and $O(\lambda \log^2 N)$ proof size and verifier time.

Our starting point is a recent work of Li et al. (CRYPTO 2026), which shows that QA codes have fast encoding and strong concrete distance. This opens the door to building efficient PCSs from QA codes via Brakedown's paradigm (Golovnev et al., CRYPTO 2023). However, a direct instantiation, called QAPCS, inherits square-root proof size and verifier time, falling short of practical efficiency when N is as large as 2^{25} . We overcome this crucial limitation by showing that QA codes are essentially code-switchable. In contrast to existing code-switching arguments that utilize algebraic structures of either generator matrices or parity-check matrices, we look into QA encoding algorithms and propose an efficient encoding-oriented argument. Consequently, **Quasar** simultaneously enjoys fast proving from QAPCS, and polylogarithmic verification of BaseFold. We implement **Quasar** over the 127-bit Mersenne prime field with rate $1/2$ and 100-bit security. Under the 32-thread CPU setting, **Quasar** accelerates commitment and evaluation over BaseFold by $13.6\times$ – $20.6\times$ and $5.2\times$ – $63.8\times$, respectively. It is also $2.2\times$ – $6.0\times$ faster in commitment than Brakedown and $2.1\times$ – $4.0\times$ faster in commitment and $17.4\times$ – $237.6\times$ faster in evaluation than BrakingBase, while providing smaller proofs and faster verification than both. Compared with QAPCS, it achieves up to $4.3\times$ faster verification and $2.9\times$ smaller proofs. Moreover, we observe that QA encoding naturally exposes massive parallelism, enabling a GPU acceleration strategy that is not directly available to the other code families. Across message lengths from 2^{12} to 2^{25} , the GPU encoder is $32.9\times$ – $148.3\times$ faster than the 32-thread CPU implementation.

Table of Contents

1	Introduction	2
1.1	Our Contributions	4
1.2	Comparison to Related Work	6
2	Preliminaries	7
3	Quasar Construction	11
3.1	From QAPCS to Quasar	12
3.2	Obstacles to Existing Frameworks for QA codes	13
3.3	A QA-Tailored Code-Switching Argument	14
3.4	The Setup Phase	18
3.5	The Commit Phase	18
3.6	The Evaluation Phase	20
4	Implementation and Evaluation	21
4.1	GPU Acceleration of QA Encoding	22
4.2	Quasar Performance	23
4.3	Matrix-multiplication verification	25
A	Efficiency	31
B	Deferred Proofs	32
B.1	Completeness	34
B.2	Binding	34
B.3	Knowledge Soundness	35
C	Parameters	40
D	Comparison with Concurrent QAFold	41

1 Introduction

Polynomial commitment schemes (PCSs) [KZG10, PST13, BBB⁺18, Lee21, WTS⁺18, GPS25, BBHR18, ACFY24, ACFY25, AHIV17, BFH⁺20, BCG20, GLS⁺23, XZS22, ZCF24, GLH⁺25, BCF⁺25, NST25, QGZ26, BFMM26, GNR26] allow a prover to commit to a polynomial and later prove that the committed polynomial evaluates to a claimed value at a given point, with a proof much smaller than an explicit description of the polynomial. This primitive is one of the main building blocks of succinct non-interactive arguments of knowledge (SNARKs), which allow a prover to convince a verifier of the correctness of a computation using a short proof. After arithmetization reduces a computation to polynomial identities and evaluation claims, the PCS is used to commit to the resulting polynomials and prove their evaluations succinctly. Field agnosticism is especially important in this setting, as it allows SNARKs to work over the application’s native field and avoids the overhead of emulating arithmetic over one field inside another. The efficiency of the PCS directly affects the prover time, verifier time, and proof size of the resulting SNARK.

A lot of recent works have focused on transparent, plausibly post-quantum PCSs based on error-correcting codes [BBHR18, ACFY24, ACFY25, AHIV17, GLS⁺23, XZS22, ZCF24, BCF⁺25, NST25, QGZ26, BFMM26, GNR26]. At a high level, existing code-based PCS constructions can be grouped into three broad approaches.

Foldable codes. A line of works builds PCSs from foldable codes, such as Reed–Solomon codes and more generally, foldable codes [BBHR18, ACFY24, ACFY25, ZCF24]. These constructions feature polylogarithmic proof size and verifier time. Classic Reed–Solomon based constructions require FFT-friendly fields due to their FFT-based encoding algorithms. This restriction is removed by BaseFold [ZCF24], which generalizes the approach to field-agnostic foldable codes with a divide-and-conquer encoding algorithm. The main drawback lies in prover time: the encoding algorithms require a quasi-linear number of multiplications over the underlying large field, which constitutes a major bottleneck in practice.

Fast-encodable codes. Another line of works, following the blueprint of Brakedown [GLS⁺23], builds PCSs from linear codes with concretely fast encoders [GLS⁺23, BFK⁺24, AHIV17, QGZ26]. For example, Brakedown uses Spielman’s linear-time encodable code [Spi95]. However, this approach has two inherent limitations. First, these codes usually have weaker provable distance guarantees. For instance, Brakedown’s encoder only achieves relative distance 0.1 at rate 1/2 under the state-of-the-art analysis [Hab23]. Second, the Brakedown framework leads to square-root proof size and verifier time. As a result, these schemes typically have larger proof size and verifier time than PCSs based on foldable codes.

Code-switching. The third approach is code-switching [NST25, BCF⁺25, XZS22, BFMM26, GNR26, BMMS25, RZR24]. The general idea is to combine a prover-efficient scheme based on a fast-encodable code with a verifier-friendly scheme based on a foldable code, aiming to inherit the fast prover of Brakedown-style constructions and the polylogarithmic proof size and verifier time of foldable-code constructions. As formalized in [BMMS25], tensor codes and LDPC codes are known examples, thanks to a tensor-product structure in the generating matrix and sparsity in the parity-check matrix, respectively. Compared with the Brakedown-style framework, the price is an additional prover cost for certifying the code-switching relation, and the main challenge is to turn the underlying code structure into efficiently checkable algebraic constraints.

QA codes and the motivation for this work. Recently, Quasi-Abelian (QA) codes have been shown to provide an attractive combination of fast encoding and strong concrete distance [LLX⁺26b, LLX⁺26a]. As shown in [LLX⁺26b], random QA codes achieve concrete relative distances close to the Gilbert–Varshamov bound. QA codes are also concretely efficient to encode. At rate 1/2, QA encoding uses only N field multiplications on a message of length N . Although it needs $2N \log N$ field additions, field multiplications dominate the cost over the large fields. As a result, QA encoding can be up to about $3.6\times$ faster than Brakedown’s

Table 1: Concrete committing cost of code-based PCSs.

Code	Rate	Mults	Permutations	Additions	Hash calls
QA [LLX ⁺ 26b]	1/2	N	0	$2N \log N$	$4N - 1$
Brakedown [GLS ⁺ 23]	1/2	$34.6N$	0	$\approx 34.6N$	$4N - 1$
Juxtaposed EA [BFK ⁺ 24]	1/4	$36N \log 2N$	0	$36N \log 2N + 4N$	$8N - 1$
RAA [BCF ⁺ 25]	1/4	0	$4N \times 2$	$8N - 2$	$8N - 1$

encoder in the same setting. Table 1 summarizes the concrete commitment cost of direct code-based PCS instantiations³

Building on these properties, [LLX⁺26b] instantiates a direct Brakedown-style PCS with QA codes, which we call QAPCS. While QAPCS already demonstrates the concrete prover-side benefits of QA encoding, it inherits the square-root proof size and verifier time of the Brakedown framework. For large polynomial sizes, although the stronger distance of QA codes reduces the number of queried columns, each opened column still has square-root length.

Therefore, our goal is to reduce the proof size and verifier time of QAPCS from square-root to polylogarithmic while preserving the concrete prover advantage of QA encoding. At first glance, one might ask whether QA codes can be made directly foldable, as in BaseFold-style PCSs. However, QA codes do not appear to have the recursive structure required by existing folding frameworks. This leads us to code switching. Existing general techniques primarily exploit either tensor decomposability of the generator matrix [BMMS25] or sparsity of the parity-check matrix [NST25]. QA codes provide neither structure in the form required by these techniques. Moreover, treating the QA encoder as a general arithmetic circuit or verifying its stages separately would introduce substantial proving overhead. The challenge is therefore to certify the complete QA encoding relation succinctly, without losing its concrete prover advantage.

This motivates the following question:

Can we design a QA-based field-agnostic PCS with concretely fast commitment, linear evaluation, and polylogarithmic proof size and verifier time?

1.1 Our Contributions

We answer the above question affirmatively by introducing **Quasar**, a field-agnostic PCS that combines QA codes with BaseFold through an efficient QA-tailored code-switching argument. For a polynomial of length N , **Quasar** achieves concretely fast $O(N \log N)$ commitment, $O(N)$ evaluation time, and $O(\lambda \log^2 N)$ proof size

³ For Juxtaposed EA and RAA, existing distance analyses for PCS-style instantiations only support inverse rate at least 4; in contrast, both QA and Brakedown’s encoder support inverse rate 2.

Table 2: Asymptotic comparison of representative code-based PCS constructions. Here N denotes the polynomial size.

Scheme	Field-agnostic	Commit time	Eval time	Verifier time	Proof size
FRI [BBHR18]	No	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
Ligero [AHIV17]	No	$O(N \log N)$	$O(N)$	$O(\sqrt{\lambda N})$	$O(\sqrt{\lambda N})$
BaseFold [ZCF24]	Yes	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
Brakedown [GLS ⁺ 23]	Yes	$O(N)$	$O(N)$	$O(\sqrt{\lambda N})$	$O(\sqrt{\lambda N})$
Blaze [BCF ⁺ 25]	No	$O(N)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
QAPCS [LLX ⁺ 26b]	Yes	$O(N \log N)$	$O(N)$	$O(\sqrt{\lambda N})$	$O(\sqrt{\lambda N})$
BrakingBase [NST25]	Yes	$O(N)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
Quasar (Ours)	Yes	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$

and verifier time. Our construction applies to arbitrary sufficiently large finite fields of odd characteristic and does not require FFT-friendly fields⁴.

A QA-tailored code-switching argument. Our main technical contribution is a direct and efficient method for certifying QA encoding. The argument exploits the specific structure of the QA encoding procedure. Intuitively, we first decompose QA encoding into a small number of structured linear and pointwise operations, and then verify each operation separately. The key observation is that, although the public matrices involved in these linear operations are not sparse, their multilinear extensions can be evaluated locally by the verifier with succinct cost. This allows each resulting consistency condition to be expressed as a sumcheck relation with succinct verifier work. We further introduce two optimizations: we exploit the shared first step across all QA output blocks and compose the encoding relations through a two-layer GKR-style reduction, thereby eliminating commitments to intermediate vectors. Consequently, our code-switching argument consists of only two sumchecks, avoiding heavy permutation checks [BCF⁺25], Spark-style commitments [Set20] to large sparse matrices [NST25], and general-purpose SNARKs [XZS22], all of which would introduce substantial prover overhead.

The Quasar PCS and concrete evaluation. Building on the above code-switching argument, we construct Quasar by combining the fast QA-code commitment with the succinct BaseFold PCS. This construction retains the concrete prover efficiency of QA encoding while reducing proof size and verifier time from the square-root complexity of QAPCS to $O(\lambda \log^2 N)$. Table 2 summarizes the high-level tradeoffs of representative code-based PCS constructions. We implement Quasar over the 127-bit Mersenne prime field and evaluate polynomial sizes from 2^{20} to 2^{29} at rate 1/2 and 100-bit security. Compared with BaseFold, Quasar is $13.6\times$ – $20.6\times$ faster in commitment and $5.2\times$ – $63.8\times$ faster in evaluation

⁴ We use BaseFold rather than FRI [BBHR18], STIR [ACFY24], or WHIR [ACFY25] in order to preserve field agnosticism, since those schemes rely on Reed-Solomon codes over FFT-friendly fields.

over the entire tested range. Its verifier time and proof size remain comparable. Compared with Brakedown, **Quasar** is $2.2\times$ – $6.0\times$ faster in commitment and $1.8\times$ – $14.8\times$ faster in verification over 2^{20} – 2^{29} . It produces $1.1\times$ – $6.3\times$ smaller proofs over 2^{21} – 2^{29} . Moreover, despite the additional code-switching checks, **Quasar** is also $1.1\times$ – $2.3\times$ faster in evaluation proving over 2^{25} – 2^{29} . Compared with QAPCS, **Quasar** is $1.3\times$ – $4.3\times$ faster in verification over 2^{24} – 2^{29} and produces $1.2\times$ – $2.9\times$ smaller proofs over 2^{25} – 2^{29} . Compared with Brakingbase, **Quasar** is $2.1\times$ – $4.0\times$ faster in commitment and $17.4\times$ – $237.6\times$ faster in evaluation. **Quasar** also achieves a $2.1\times$ – $3.2\times$ faster verifier and produces $2.2\times$ – $3.5\times$ smaller proofs.

GPU-accelerated QA encoding and commitment. We observe that QA encoding is suitable with GPU acceleration and develop a GPU implementation that directly maps the QA encoding to parallel GPU execution. We further integrate this encoder into a hybrid GPU-CPU commitment pipeline: the GPU performs QA encoding and column hashing and returns only the Merkle leaf digests, whereas the CPU constructs the remaining Merkle tree and executes the remaining proof. Across message lengths 2^{12} – 2^{25} , the GPU encoder achieves an $8.8\times$ – $203.1\times$ speedup over the serial CPU implementation and a $32.9\times$ – $148.3\times$ speedup over the 32-thread CPU implementation. For polynomial sizes 2^{20} – 2^{29} , the hybrid pipeline accelerates the commitment phase by $3.4\times$ – $16.7\times$ relative to the 32-thread CPU implementation, without changing the commitment, proof, or verifier algorithm.

1.2 Comparison to Related Work

Orion. Orion [XZS22] starts with Brakedown’s encoder and performs code-switching via Virgo [ZXZS20], a general-purpose SNARK for arithmetic circuits. Since Virgo relies on FFT-based techniques, Orion is not field-agnostic. Notably, Orion uses the SPARK compiler [Set20] to commit to the wiring predicates required by the GKR protocol [GKR08], which introduces substantial prover overhead. In contrast, **Quasar** avoids invoking a general-purpose SNARK or a SPARK-style compiler for the switching step, and proves the code-switching relation directly using a QA-tailored algebraic argument.

Juxtaposed Expand-Accumulate codes. Juxtaposed EA codes were analyzed in detail over arbitrary large fields in [BFK⁺24]. Their encoders run in quasilinear time and are concretely quite efficient, while their provable distance guarantees are comparable to those of the Brakedown code. In comparison, QA codes offer stronger concrete distance in our target regimes and an encoder with only linear multiplication complexity, leading to better concrete encoding efficiency.

Blaze. Blaze [BCF⁺25] is a code-switching PCS that combines RAA codes with BaseFold. However, current distance analyses of RAA codes are restricted to binary fields, thus Blaze has not been extended to large prime fields yet. Moreover, their distance analysis requires inverse rate at least 4, whereas QA codes in our target regime support inverse rate 2 with stronger concrete distance guarantees. This leads to smaller committed codewords and faster commitments in **Quasar**.

The encoding check is also cleaner in our setting: the QA encoding relation admits a simple algebraic decomposition that can be certified directly by sumcheck-style checks, whereas **Blaze** needs more involved permutation checks [CBBZ23] for the RAA encoding. RAA codes also impose more restrictive message-length requirements.

BrakingBase. BrakingBase [NST25] combines Brakedown’s encoder with BaseFold to realize polylogarithmic proof size and verifier time. The main difference to Quasar is that we use QA codes, whose encoding is concretely faster in our target regimes. Another difference is the cost of checking the switching relation. BrakingBase verifies this relation through parity-check-matrix constraints, which invokes the SPARK compiler and introduces a substantial prover overhead. In contrast, Quasar is built around a QA-tailored switching argument, avoiding this SPARK-based certification step. This makes our proof more direct and lightweight.

Concurrent works. Concurrent works such as Bolt [GNR26], ERA codes [BFMM26], and Lightning [QGZ26] study another direction for constructing efficient code-based PCSs. These works start from a base code and apply additional transformations to improve concrete efficiency. ERA codes achieve a strong tradeoff between encoding time and relative distance: they obtain commitment time close to that of Brakedown while improving the concrete distance. Meanwhile, Bolt and Lightning are heavily optimized for prover time, but the relative distance of the resulting codes is typically in the range of roughly 0.001 to 0.01. More recently, SwitchFold [CYY26] develops a code-agnostic PCS framework. Its instantiation with QA codes, called QAFold, also uses QA codes but follows a fundamentally different switching strategy. SwitchFold treats the underlying code generically and obtains succinctness through recursive code switching, whereas Quasar directly exploits the internal structure of QA encoding and only needs two sumcheck. Thus, SwitchFold emphasizes code agnosticity and applies to a broad class of linear codes, while Quasar trades this generality for a concretely more efficient QA-tailored scheme. Detailed comparison is in Appendix D.

2 Preliminaries

Notation. Let $[n]$ denote the set $\{1, \dots, n\}$ where n is a positive integer. We use bold lowercase letters, such as $\mathbf{x}, \mathbf{y}, \mathbf{z}$, to denote column vectors, and bold uppercase letters, such as $\mathbf{M}, \mathbf{C}$, to denote matrices. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}^n$, we write $\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{j=1}^n x_j y_j$ for their standard inner product, and write $\mathbf{x} \star \mathbf{y} := (x_1 y_1, \dots, x_n y_n)^\top$ for their component-wise product. For a matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$, we write $\mathbf{A}(i, \cdot) \in \mathbb{F}^n$ to denote the i -th row of $\mathbf{A}$, and $\mathbf{A}(\cdot, j) \in \mathbb{F}^m$ to denote the j -th column of $\mathbf{A}$. For an integer $i \in \{0, \dots, 2^k - 1\}$, we write $\langle i \rangle \in \{0, 1\}^k$ for its bit binary representation.

Multilinear extensions. Every function $f : \{0, 1\}^k \rightarrow \mathbb{F}$ has a unique multilinear extension, denoted by $\tilde{f} : \mathbb{F}^k \rightarrow \mathbb{F}$. For $\mathbf{z} \in \mathbb{F}^k$, it can be written as

$$\tilde{f}(\mathbf{z}) = \sum_{\mathbf{b} \in \{0, 1\}^k} \tilde{\text{eq}}(\mathbf{z}, \mathbf{b}) f(\mathbf{b}),$$

where

$$\tilde{\text{eq}}(\mathbf{z}, \mathbf{b}) = \prod_{i=1}^k (z_i b_i + (1 - z_i)(1 - b_i)).$$

For a vector $\mathbf{v} \in \mathbb{F}^N$ with $N = 2^k$, we identify $\mathbf{v}$ with the function on $\{0, 1\}^k$ defined by $\mathbf{v}(\langle i \rangle) = v_i, i \in \{0, \dots, N - 1\}$, and denote its multilinear extension by $\tilde{v}$.

Sumcheck. We recall the sumcheck protocol of Lund et al. [LFKN90]. Let $Q : \mathbb{F}^m \rightarrow \mathbb{F}$ be an m -variate polynomial with individual degree at most d . The verifier wants to check a claimed identity $y = \sum_{\mathbf{x} \in \{0, 1\}^m} Q(\mathbf{x})$. The sumcheck protocol reduces this exponentially large sum to one evaluation of Q at a random point. The protocol proceeds in m rounds. In round i , for $i = 1, \dots, m$, the prover sends the univariate polynomial

$$g_i(X) := \sum_{\mathbf{b} \in \{0, 1\}^{m-i}} Q(r_1, \dots, r_{i-1}, X, \mathbf{b}),$$

where $r_1, \dots, r_{i-1}$ are the verifier's previous challenges. For $i = 1$, the verifier checks $g_1(0) + g_1(1) = y$. For $i > 1$, it checks $g_i(0) + g_i(1) = g_{i-1}(r_{i-1})$. If the check passes, the verifier samples a fresh challenge $r_i \leftarrow \mathbb{F}$. At the end, the verifier queries Q at $\mathbf{r} = (r_1, \dots, r_m)$ and accepts if $Q(\mathbf{r}) = g_m(r_m)$.

The protocol is perfectly complete. If the claimed sum is false, then any cheating prover can convince the verifier with probability at most $\frac{md}{|\mathbb{F}|}$. In each round, the prover sends $d + 1$ field elements and the verifier responds with one field element. The verifier performs $O(dm)$ field operations. Using the linear-time prover of [Tha13], the prover runs in time linear in the size of the evaluation table. In particular, when Q is a product of d multilinear polynomials over $\{0, 1\}^m$, the prover time is $O(d \cdot 2^m)$.

Batched sumcheck. We also use the standard batching technique for sumcheck claims. Suppose the verifier wants to check k identities $y_j = \sum_{\mathbf{x} \in \{0, 1\}^m} Q_j(\mathbf{x})$, $j = 0, \dots, k - 1$, where each $Q_j : \mathbb{F}^m \rightarrow \mathbb{F}$ has individual degree at most d . Instead of running k separate sumcheck protocols, the verifier samples a random challenge $\alpha \leftarrow \mathbb{F}$ and defines

$$Q(\mathbf{x}) := \sum_{j=0}^{k-1} \alpha^j Q_j(\mathbf{x}), \quad y := \sum_{j=0}^{k-1} \alpha^j y_j.$$

The parties then run a single sumcheck protocol for $y = \sum_{\mathbf{x} \in \{0, 1\}^m} Q(\mathbf{x})$. If all original claims are true, then the batched claim is true. If at least one claim is

false, then the random linear combination masks the error with probability at most $(k-1)/|\mathbb{F}|$. Thus, the total soundness error is at most $\frac{k-1}{|\mathbb{F}|} + \frac{md}{|\mathbb{F}|}$.

Interactive oracle proofs and arguments of knowledge. Let $\mathcal{R}$ be an NP relation and let $L_{\mathcal{R}} := \{x : \exists w \text{ such that } (x, w) \in \mathcal{R}\}$ be the corresponding language. An interactive proof for $\mathcal{R}$ allows a prover to convince a verifier that, for a public instance x , it knows a witness w such that $(x, w) \in \mathcal{R}$. We write $\langle P, V \rangle(x; w)$ for the output of the verifier after interacting with prover P on public input x , where the witness w is known only to the prover.

An interactive oracle proof, or IOP, is an interactive proof in which the prover's messages are given as oracles. In a polynomial IOP, these oracles are low-degree polynomials, and the verifier accesses them only through a small number of oracle queries. Thus, instead of reading the prover's entire messages, the verifier queries the submitted polynomial oracles at selected points.

Definition 2.1 (Argument of knowledge). *Let $\mathcal{R}$ be an NP relation. An argument of knowledge for $\mathcal{R}$ consists of a setup algorithm Gen and a pair of interactive algorithms (P, V) satisfying the following properties.*

- *Completeness: For every security parameter $\lambda \in \mathbb{N}$, every public parameter $\text{pp} \leftarrow \text{Gen}(1^\lambda)$, and every $(x, w) \in \mathcal{R}$, we have*

$$\Pr[\langle P, V \rangle(\text{pp}, x; w) = 1] = 1.$$

- *Knowledge soundness: For every PPT adversarial prover P^* , there exists a PPT extractor Ext^{P^*} such that*

$$\Pr \left[\begin{array}{c} \langle P^*, V \rangle(\text{pp}, x) = 1 \\ \wedge (x, w) \notin \mathcal{R} \end{array} \mid \begin{array}{c} \text{pp} \leftarrow \text{Gen}(1^\lambda), \\ x \leftarrow P^*(\text{pp}), \\ w \leftarrow \text{Ext}^{P^*}(\text{pp}, x) \end{array} \right] \leq \text{negl}(\lambda).$$

Polynomial Commitment Scheme. A polynomial commitment scheme for multilinear polynomials is a tuple of four protocols $\text{PC} = (\text{Gen}, \text{Commit}, \text{Open}, \text{Eval})$ defined as follows.

- $\text{pp} \leftarrow \text{Gen}(1^\lambda, \mu)$: takes as input the security parameter λ and the number of variables μ , and outputs public parameters pp .
- $(c, h) \leftarrow \text{Commit}(\text{pp}, f)$: takes as input a μ -variate multilinear polynomial $f \in \mathbb{F}^{\leq 1}[X_1, \dots, X_\mu]$ and outputs a public commitment c together with private auxiliary opening information h . The value h is kept by the committer and may be used later as an opening hint.
- $b \leftarrow \text{Open}(\text{pp}, c, f, h)$: takes as input a commitment c , a μ -variate multilinear polynomial f , and an opening hint h . It outputs 1 if h certifies that c opens to f , and outputs 0 otherwise.
- $b \leftarrow \text{Eval}(\text{pp}, c, z, y, \mu, f)$: is an interactive protocol between a PPT prover $\mathcal{P}$ and a verifier $\mathcal{V}$. Both parties hold the commitment c , the number of variables μ , a point $z \in \mathbb{F}^\mu$, and a claimed value $y \in \mathbb{F}$. The prover additionally knows a μ -variate multilinear polynomial f . The prover attempts to convince the verifier that $f(z) = y$ and that this claim is consistent with the commitment c . At the end of the protocol, the verifier outputs a bit $b \in \{0, 1\}$.

$\text{PC} = (\text{Gen}, \text{Commit}, \text{Open}, \text{Eval})$ is an extractable polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$ if the following conditions hold.

Definition 2.2 (Completeness). *For any μ -variate multilinear polynomial f , for any point $\mathbf{z} \in \mathbb{F}^\mu$, and for $y = f(\mathbf{z})$, we have*

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Gen}(1^\lambda, \mu); \\ (\mathbf{c}, \mathbf{h}) \leftarrow \text{Commit}(\text{pp}, f); \\ \text{Eval}(\text{pp}, \mathbf{c}, \mathbf{z}, y, \mu, f) = 1 \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Definition 2.3 (Binding). *For any PPT adversary $\mathcal{A}$ and any size parameter $\mu \geq 1$,*

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Gen}(1^\lambda, \mu); \\ (\mathbf{c}, f_0, \mathbf{h}_0, f_1, \mathbf{h}_1) \leftarrow \mathcal{A}(\text{pp}); b_0 = b_1 = 1 \\ b_0 \leftarrow \text{Open}(\text{pp}, \mathbf{c}, f_0, \mathbf{h}_0); \quad \wedge \quad f_0 \neq f_1 \\ b_1 \leftarrow \text{Open}(\text{pp}, \mathbf{c}, f_1, \mathbf{h}_1) : \end{array} \right] \leq \text{negl}(\lambda).$$

Definition 2.4 (Knowledge soundness). *The protocol Eval is a succinct argument of knowledge for the NP relation defined, for $\text{pp} \leftarrow \text{Gen}(1^\lambda, \mu)$, as*

$$\mathcal{R}_{\text{Eval}}(\text{pp}) = \left\{ (\mathbf{c}, \mathbf{z}, y; f) : \begin{array}{l} f \in \mathbb{F}^{\leq 1}[X_1, \dots, X_\mu], \\ f(\mathbf{z}) = y, \\ \exists \mathbf{h}, \text{Open}(\text{pp}, \mathbf{c}, f, \mathbf{h}) = 1 \end{array} \right\}.$$

Foldable codes and BaseFold [ZCF24]. We use the BaseFold PCS as a black-box multilinear polynomial commitment scheme. At a high level, BaseFold is built from a family of foldable linear codes. Let ρ denote the inverse rate. Starting from a base generator matrix $\mathbf{G}_0 \in \mathbb{F}^{k_0 \times \rho k_0}$, the generator matrices are defined recursively. For $k_i = k_0 \cdot 2^i$, the matrix $\mathbf{G}_i \in \mathbb{F}^{k_i \times \rho k_i}$ is obtained from $\mathbf{G}_{i-1}$ by sampling a diagonal matrix $\mathbf{T}_{i-1}$ and setting

$$\mathbf{G}_i = \begin{pmatrix} \mathbf{G}_{i-1} & \mathbf{G}_{i-1} \\ \mathbf{G}_{i-1}\mathbf{T}_{i-1} & -\mathbf{G}_{i-1}\mathbf{T}_{i-1} \end{pmatrix}.$$

This recursive structure allows a codeword of length ρk_i to be folded into a shorter codeword while preserving consistency with a folded message.

The BaseFold commitment algorithm commits to a vector $\mathbf{m} \in \mathbb{F}^K$ by encoding it with the foldable code generated by $\mathbf{G}_i$ and building a Merkle tree over the resulting codeword. The evaluation protocol proves a claim of the form $\tilde{m}(\mathbf{r}) = v$, where $\tilde{m}$ is the multilinear extension of the committed vector $\mathbf{m} \in \mathbb{F}^K$. The claim is first expressed as $v = \sum_{\mathbf{b} \in \{0,1\}^{\log K}} \tilde{\mathbf{e}}\mathbf{q}(\mathbf{r}, \mathbf{b}) m(\mathbf{b})$. This sumcheck is run in parallel with the recursive folding process of BaseFold, using the same verifier challenges in both procedures. For a vector of length K and constant inverse rate ρ , BaseFold has commitment time $O(K \log K)$ and evaluation-prover time $O(K)$. Its verifier time and proof size are both $O(\lambda \log^2 K)$ for security parameter λ . These are the black-box properties used by Quasar.

We also use the batch evaluation protocol of BaseFold, following the standard batching method used in [NST25]. Suppose the verifier is given commitments $\mathbf{c}_1, \dots, \mathbf{c}_k$ to multilinear polynomials $\tilde{f}_1, \dots, \tilde{f}_k : \mathbb{F}^\ell \rightarrow \mathbb{F}$, and wants to verify the claims $\tilde{f}_t(\boldsymbol{\alpha}_t) = y_t, t \in [k]$. Instead of proving these claims separately, the verifier samples random challenges and reduces them to a single sumcheck claim:

$$\sum_{t \in [k]} \gamma_t y_t = \sum_{\mathbf{x} \in \{0,1\}^\ell} \left(\sum_{t \in [k]} \gamma_t \tilde{\text{eq}}(\boldsymbol{\alpha}_t, \mathbf{x}) f_t(\mathbf{x}) \right).$$

At the end of the sumcheck protocol, the verifier obtains a random point $\boldsymbol{\zeta} \in \mathbb{F}^\ell$ and needs to check the evaluations $\tilde{f}_t(\boldsymbol{\zeta}) = \theta_t, t \in [k]$. These remaining claims are further batched by another random linear combination. Namely, the verifier samples random challenges $\eta_1, \dots, \eta_k \leftarrow \mathbb{F}$, and the prover proves the single claim $\sum_{t \in [k]} \eta_t \tilde{f}_t(\boldsymbol{\zeta}) = \sum_{t \in [k]} \eta_t \theta_t$ using one BaseFold evaluation proof for the virtual linear combination of the committed polynomials.

For k evaluation claims, the protocol uses one ℓ -round sumcheck and one batched BaseFold evaluation proof. When k is constant, the prover time remains linear in 2^ℓ , and the verifier time and proof size remain polylogarithmic in 2^ℓ .

Quasi-Abelian codes. We briefly recall the systematic QA codes used in this work. Let $\mathbb{F}$ be a finite field of odd characteristic, and let $N = 2^n$. A rank-1 systematic QA code of index c and message length N is specified by a generator matrix

$$\mathbf{G}_n = (\mathbf{I}_N \mid \mathbf{A}_{n,2} \mid \dots \mid \mathbf{A}_{n,c}) \in \mathbb{F}^{N \times cN},$$

where each $\mathbf{A}_{n,i} \in \mathbb{F}^{N \times N}$ is sampled independently from the following recursive distribution. For level 0, sample a random scalar $\xi \leftarrow \mathbb{F}$ and set

$$\mathbf{A}_0 = [\xi] \in \mathbb{F}^{1 \times 1}.$$

For level $\ell \geq 1$, independently sample two matrices $\mathbf{A}_{\ell-1,L}, \mathbf{A}_{\ell-1,R} \in \mathbb{F}^{2^{\ell-1} \times 2^{\ell-1}}$ from the level- $(\ell-1)$ distribution, and set

$$\mathbf{A}_\ell = \begin{pmatrix} \mathbf{A}_{\ell-1,L} & \mathbf{A}_{\ell-1,R} \\ \mathbf{A}_{\ell-1,R} & \mathbf{A}_{\ell-1,L} \end{pmatrix} \in \mathbb{F}^{2^\ell \times 2^\ell}.$$

Each matrix $\mathbf{A}_{n,i}$ is sampled independently according to this level- n distribution.

The resulting code is $\mathcal{C}_{\text{QA}} := \{\mathbf{m} \mathbf{G}_n : \mathbf{m} \in \mathbb{F}^N\} \subseteq \mathbb{F}^{cN}$. It is a linear code of message length N and codeword length cN , with rate $1/c$. We denote its relative distance by δ_{QA} , whose concrete lower bound follows from [LLX⁺26b, Theorem 4.4 and Corollary 3.21].

3 Quasar Construction

In this section, we present the construction of Quasar. Section 3.1 first reviews the direct Brakedown-style PCS based on QA codes, which we call QAPCS, and

explains how the code switching approach works. Section 3.2 then explains why QA codes do not directly fit existing foldable-code, tensor-code, or LDPC-code frameworks. In Section 3.3, we show that the QA encoding admits sumcheck representations. Then we give two optimizations to compose these relations into a two-layer reduction that eliminates the commitments to intermediate encoding vectors. Finally, Sections 3.4, 3.5, and 3.6 instantiate the setup, commitment, and evaluation protocols.

3.1 From QAPCS to Quasar

We begin by recalling the direct Brakedown-style instantiation with QA codes, which we call QAPCS.

QAPCS Commit Phase. Suppose that the prover commits to a multilinear polynomial $f(X_1, \dots, X_n) \in \mathbb{F}[X_1, \dots, X_n]$. The prover represents its evaluations over the Boolean hypercube $\{0, 1\}^n$ as a matrix $\mathbf{M} \in \mathbb{F}^{N_1 \times N_2}$, where $N = 2^n = N_1 \times N_2$ and $N_1 = 2^{n_1}, N_2 = 2^{n_2}$. The entry of $\mathbf{M}$ at row $\mathbf{i}$ and column $\mathbf{j}$, where $\mathbf{i} \in \{0, 1\}^{n_1}$, and $\mathbf{j} \in \{0, 1\}^{n_2}$, corresponds to the evaluation of f at the point $(\mathbf{i}, \mathbf{j})$. Then the prover encodes each row of $\mathbf{M}$ by QA.Enc and obtains a matrix $\mathbf{C} \in \mathbb{F}^{N_1 \times cN_2}$, where c is the index of QA codes. The commitment to f is the root of the Merkle tree built over the columns of $\mathbf{C}$.

QAPCS Proximity Test. In the proximity test, the verifier samples a random $\mathbf{r} \in \mathbb{F}^{N_1}$ and sends it to the prover. The prover computes $\mathbf{v} = \mathbf{r}^\top \cdot \mathbf{M}$ and sends the result to the verifier. To check the correctness of this computation, the verifier sends a random set $I \subset \{0, \dots, c \cdot N_2 - 1\}$ of size $\Theta(\lambda)$ to the prover. The prover then opens the columns $\mathbf{C}(\cdot, i)$ for all $i \in I$. The verifier encodes $\mathbf{v}$ via QA.Enc, obtains a codeword $\mathbf{u}$, and checks the validity of all the corresponding Merkle-tree proof paths of $\mathbf{C}(\cdot, i)$ and $u_i = \langle \mathbf{r}, \mathbf{C}(\cdot, i) \rangle$ for all $i \in I$. If all checks pass, the verifier is convinced that all rows of $\mathbf{C}$ are close to QA codewords with high probability.

QAPCS Evaluation Check. The evaluation check is quite similar to the proximity test. Let the claimed evaluation be $f(\mathbf{z}) = y$, where $\mathbf{z} = (z_1, \dots, z_n)$. Define that $\mathbf{z}_L = \bigotimes_{i=1}^{n_1} (1 - z_i, z_i) \in \mathbb{F}^{N_1}$, $\mathbf{z}_R = \bigotimes_{i=n_1+1}^n (1 - z_i, z_i) \in \mathbb{F}^{N_2}$. Then the evaluation relation can be rewritten as $f(\mathbf{z}) = \mathbf{z}_L^\top \mathbf{M} \mathbf{z}_R$. Therefore, verifying $f(\mathbf{z}) = y$ reduces to checking the matrix inner-product relation $y = \langle \mathbf{p}, \mathbf{z}_R \rangle$, where $\mathbf{p} = \mathbf{z}_L^\top \mathbf{M}$. Thus, $\mathbf{z}_L$ plays the same role as $\mathbf{r}$ in the proximity test: the prover sends $\mathbf{p} = \mathbf{z}_L^\top \mathbf{M}$, and the remaining checks ensure that this vector is consistent with the committed matrix $\mathbf{M}$. At the end of the evaluation check, the verifier accepts only if $\langle \mathbf{p}, \mathbf{z}_R \rangle = y$.

The proof contains the two row combination vectors and the authenticated columns. Its size is therefore $O(N_2 + \lambda N_1)$. Balancing the two terms gives an optimal proof size of $O(\sqrt{\lambda N})$. This is the main bottleneck of QAPCS. Our goal in Quasar is to retain the fast commitment of QA codes while reducing the proof size and verifier time to $O(\lambda \log^2 N)$.

Code-Switching. We overview how the code-switching approach works. We focus on the proximity part of the evaluation protocol; the evaluation check is analogous. The code-switching approach avoids sending the long vector $\mathbf{v} = \mathbf{r}^\top \mathbf{M} \in \mathbb{F}^{N_2}$ directly. Instead, the prover computes the QA codeword $\mathbf{u} = \text{QA.Enc}(\mathbf{v})$, partitions it as $\mathbf{u} = (\mathbf{u}_1 \parallel \cdots \parallel \mathbf{u}_c)$, and commits to the blocks $\mathbf{u}_1, \dots, \mathbf{u}_c$ using BaseFold. Since the QA code is systematic, $\mathbf{u}_1 = \mathbf{v}$.

The verifier needs to check two conditions:

1. $\mathbf{u}$ is consistent with the original committed matrix $\mathbf{C}$ on the sampled columns;
2. $\mathbf{u}$ is a valid QA codeword.

For Item 1, we follow the standard technique to batch multiple checks into a single sumcheck claim. Item 2 is the actual code-switching check. We will elaborate on this subsequently.

The actual evaluation check uses the same structure with $\mathbf{z}_L$ in place of $\mathbf{r}$. Namely, the prover computes $\mathbf{p} = \mathbf{z}_L^\top \mathbf{M}$, commits to $\mathbf{q} = \text{QA.Enc}(\mathbf{p})$ using BaseFold, and proves both that $\mathbf{q}$ is consistent with the sampled columns of $\mathbf{C}$ and that $\mathbf{q}$ is a valid QA codeword. Finally, since the QA code is systematic, the first block satisfies $\mathbf{q}_1 = \mathbf{p}$, and the verifier checks the final evaluation claim $\tilde{q}_1(\mathbf{z}_R) = y$ through the batched BaseFold evaluation protocol.

By setting $N_1 = \Theta(\log N)$, $N_2 = N/N_1$, the opened columns have height N_1 and contribute $O(\lambda \log N)$ to the proof size. As shown in Appendix A, all remaining claims are reduced to batched BaseFold evaluations, giving $O(\lambda \log^2 N)$ verifier time and proof size. The additional prover work remains linear because the BaseFold commitments are over vectors of length $N_2 = N/\Theta(\log N)$.

3.2 Obstacles to Existing Frameworks for QA codes

In this subsection, we explain why QA codes do not directly fit existing foldable-code, tensor-code, or LDPC-code switching frameworks.

From the generating-matrix viewpoint, a systematic QA encoder has the form

$$\mathbf{v} \mapsto (\mathbf{v}, \mathbf{v}\mathbf{A}_2, \dots, \mathbf{v}\mathbf{A}_c),$$

where the matrices $\mathbf{A}_2, \dots, \mathbf{A}_c$ are dense structured matrices. Although the matrices $\mathbf{A}_n$ have a recursive block structure, this structure is not the folding structure required by foldable-code frameworks such as BaseFold. To see this, consider the rate-1/2 case with generating matrix

$$\mathbf{G}_n = (\mathbf{I}_N \mid \mathbf{A}_n), \quad \mathbf{A}_n = \begin{pmatrix} \mathbf{A}_L & \mathbf{A}_R \\ \mathbf{A}_R & \mathbf{A}_L \end{pmatrix}.$$

For a message $\mathbf{m} = (\mathbf{m}_0, \mathbf{m}_1)$, the corresponding codeword is $\mathbf{m}\mathbf{G}_n = (\mathbf{m}_0, \mathbf{m}_1, \mathbf{p}_0, \mathbf{p}_1)$, where

$$\mathbf{p}_0 = \mathbf{m}_0\mathbf{A}_L + \mathbf{m}_1\mathbf{A}_R, \quad \mathbf{p}_1 = \mathbf{m}_0\mathbf{A}_R + \mathbf{m}_1\mathbf{A}_L.$$

A natural attempt is to fold the message by setting $\mathbf{m}_\beta = \mathbf{m}_0 + \beta\mathbf{m}_1$ for a verifier challenge β , and then ask whether the right part of the folded codeword

can be obtained as a linear combination of the two original right parts. Namely, one would need $\lambda, \mu \in \mathbb{F}$ and a matrix $\mathbf{A}_{n-1}$ such that

$$\lambda \mathbf{p}_0 + \mu \mathbf{p}_1 = (\mathbf{m}_0 + \beta \mathbf{m}_1) \mathbf{A}_{n-1}.$$

By comparing the coefficients of $\mathbf{m}_0$ and $\mathbf{m}_1$, this requires

$$(\beta \lambda - \mu) \mathbf{A}_L + (\beta \mu - \lambda) \mathbf{A}_R = 0.$$

Since $\mathbf{A}_L$ and $\mathbf{A}_R$ are sampled independently, with overwhelming probability, they do not satisfy a nontrivial linear relation. For a random β with $\beta^2 \neq 1$, the above condition then forces $\lambda = \mu = 0$, yielding only the trivial combination. Hence the full random QA does not admit such a folding structure.

One may obtain the folding structure by restricting the QA matrices to a more structured subfamily. For example,

$$\mathbf{A}_n = \begin{pmatrix} \mathbf{A}_{n-1} & z_{n-1} \mathbf{A}_{n-1} \\ z_{n-1} \mathbf{A}_{n-1} & \mathbf{A}_{n-1} \end{pmatrix} = z_0 \bigotimes_{i=1}^n \begin{pmatrix} 1 & z_i \\ z_i & 1 \end{pmatrix}.$$

Thus

$$\mathbf{p}_0 = (\mathbf{m}_0 + z_{n-1} \mathbf{m}_1) \mathbf{A}_{n-1}, \quad \mathbf{p}_1 = (z_{n-1} \mathbf{m}_0 + \mathbf{m}_1) \mathbf{A}_{n-1}.$$

For the folded message $\mathbf{m}_\beta = \mathbf{m}_0 + \beta \mathbf{m}_1$, the desired shorter right part is $\mathbf{m}_\beta \mathbf{A}_{n-1}$. Indeed, by comparing coefficients, one obtains

$$(\mathbf{m}_0 + \beta \mathbf{m}_1) \mathbf{A}_{n-1} = \frac{1 - \beta z_{n-1}}{1 - z_{n-1}^2} \mathbf{p}_0 + \frac{\beta - z_{n-1}}{1 - z_{n-1}^2} \mathbf{p}_1,$$

provided $z_{n-1}^2 \neq 1$. Therefore this restricted family admits a QA-specific folding rule.

However, this restriction substantially reduces the randomness of the QA generator and destroys the distance guarantees. A full random QA matrix at level n is determined by $N = 2^n$ independent field coefficients. In contrast, the factorized family above is determined only by the scalars $z_0, z_1, \dots, z_n$, namely by $n + 1 = O(\log N)$ field elements.

Moreover, QA codes do not match the other known code-switching templates. They are clearly not tensor codes: their recursive blocks are independently sampled QA matrices, rather than tensor factors of a fixed small generator, so the tensor-code code-switching does not apply. They are also not LDPC codes: the matrices $\mathbf{A}_2, \dots, \mathbf{A}_c$ are dense, and the induced parity-check relations are not sparse. Thus, efficient QA code-switching requires a different certification strategy.

3.3 A QA-Tailored Code-Switching Argument

Recall that Item 2 is to prove that the blocks committed by BaseFold form a valid QA codeword. Our main observation is that all three stages of the QA encoder

admit sumcheck representations. Consequently, the QA encoding relation can be verified using sumchecks rather than a general-purpose circuit argument.

We obtain this by using the polynomial viewpoint of QA codes [LLX⁺26b, LLX⁺26a]. Let $\mathcal{R}_{n_2} := \mathbb{F}[X_1, \dots, X_{n_2}]/(X_1^2 - 1, \dots, X_{n_2}^2 - 1)$. A message vector $\mathbf{v} \in \mathbb{F}^{N_2}$ is identified with the coefficient vector of an element $v \in \mathcal{R}_{n_2}$, where $N_2 = 2^{n_2}$. For public elements $a_1, \dots, a_c \in \mathcal{R}_{n_2}$, a QA codeword now has the form $(va_1, \dots, va_c)$. Thus, encoding amounts to computing c multiplications in $\mathcal{R}_{n_2}$. In the systematic case, $a_1 = 1$, and the first block is the message.

This polynomial viewpoint leads to an efficient encoder. Multiplication in $\mathcal{R}_{n_2}$ can be performed by evaluating the two input polynomials on the set $\{-1, 1\}^{n_2}$, multiplying the evaluations component-wisely, and interpolating back. This is exactly the Walsh–Hadamard transform (WHT). Let $\mathbf{a}_i$ denote the coefficient vector of a_i , and let $\mathbf{a}'_i = \text{WHT}(\mathbf{a}_i)$. To avoid inverse WHT, we let the public parameters contain the scaled vectors $2^{-n_2}\mathbf{a}'_1, \dots, 2^{-n_2}\mathbf{a}'_c \in \mathbb{F}^{N_2}$. Now to encode $\mathbf{v}$, it is equivalent to perform the following three steps:

1. $\mathbf{v}' = \text{WHT}(\mathbf{v})$;
2. $\mathbf{u}'_i = 2^{-n_2}\mathbf{a}'_i \star \mathbf{v}'$ for every $i \in [c]$;
3. $\mathbf{u}_i = \text{WHT}(\mathbf{u}'_i)$ for every $i \in [c]$.

It outputs $\mathbf{u} = (\mathbf{u}_1 \parallel \dots \parallel \mathbf{u}_c)$ as the QA codeword of $\mathbf{v}$. The first block is systematic, namely $\mathbf{u}_1 = \mathbf{v}$.

Certifying the WHT layers. The first and third steps are WHT relations. $\text{WHT}(\mathbf{v})$ is precisely the evaluation vector of v over all points in $\{1, -1\}^{n_2}$. Equivalently, it can be written as

$$\text{WHT}(\mathbf{v}) = \mathbf{H}^{\otimes n_2} \mathbf{v}, \quad \mathbf{H} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

The key observation is that this tensor-product structure is useful not only for accelerating the prover’s encoding algorithm, but also for enabling succinct verification. It allows the verifier to locally evaluate the multilinear extension of the WHT matrix $\mathbf{H}^{\otimes n_2}$ in logarithmic time, and hence turns each WHT relation in the QA encoder into a sumcheck relation with small verifier overhead.

Let $\mathbf{H}_{n_2} := \mathbf{H}^{\otimes n_2}$ denote the WHT matrix of size $N_2 \times N_2$. To verify a relation of the form $\mathbf{v}' = \text{WHT}(\mathbf{v})$, the verifier samples a random point $\boldsymbol{\gamma} \in \mathbb{F}^{n_2}$, and the parties reduce the claim to

$$\tilde{v}'(\boldsymbol{\gamma}) = \sum_{\mathbf{x} \in \{0,1\}^{n_2}} \tilde{H}_{n_2}(\boldsymbol{\gamma}, \mathbf{x}) \tilde{v}(\mathbf{x}).$$

This identity can be proved by sumcheck. The terminal claim involves evaluations of $\tilde{v}$ and $\tilde{v}'$, which can be checked using the BaseFold PCS.

We now explain the local evaluation of $\tilde{H}_{n_2}$. Index the rows and columns of $\mathbf{H}$ by bits $a, b \in \{0, 1\}$. Then the entry of $\mathbf{H}$ at row a and column b is

$$\mathbf{H}(a, b) = \begin{cases} -1, & a = b = 1, \\ 1, & \text{otherwise,} \end{cases}$$

whose multilinear extension over $\{0, 1\}^2$ is exactly $\tilde{H}(a, b) = 1 - 2ab$. Since $\mathbf{H}_{n_2} = \mathbf{H}^{\otimes n_2}$, the uniqueness of multilinear extensions gives

$$\tilde{H}_{n_2}(\mathbf{a}, \mathbf{b}) = \prod_{j=1}^{n_2} (1 - 2a_j b_j), \quad \mathbf{a}, \mathbf{b} \in \mathbb{F}^{n_2}.$$

Thus the verifier can evaluate $\tilde{H}_{n_2}$ locally using $O(n_2)$ field operations⁵. The same argument verifies the third step, namely the relations $\mathbf{u}_i = \text{WHT}(\mathbf{u}'_i)$ for all $i \in [c]$.

Certifying the pointwise multiplication layer. It remains to verify the second step, which consists of pointwise multiplication relations. Instead of checking the c relations separately, we let the verifier sample a random challenge $\alpha \in \mathbb{F}$ and batche them into the identity

$$\sum_{i=1}^c \alpha^{i-1} \tilde{u}'_i(\mathbf{x}) = \sum_{i=1}^c \alpha^{i-1} 2^{-n_2} \tilde{a}'_i(\mathbf{x}) \tilde{v}'(\mathbf{x}) \quad \forall \mathbf{x} \in \{0, 1\}^{n_2}.$$

This is a standard zerocheck statement [Set20, CBBZ23]. The verifier samples $\beta \in \mathbb{F}^{n_2}$, and the parties run sumcheck on

$$\sum_{\mathbf{x} \in \{0, 1\}^{n_2}} \text{eq}(\beta, \mathbf{x}) \left(\sum_{i=1}^c \alpha^{i-1} \tilde{u}'_i(\mathbf{x}) - \sum_{i=1}^c \alpha^{i-1} 2^{-n_2} \tilde{a}'_i(\mathbf{x}) \tilde{v}'(\mathbf{x}) \right) = 0.$$

The vectors $\mathbf{a}'_1, \dots, \mathbf{a}'_c$ are public parameters of the QA code. Therefore, their BaseFold commitments can be generated once in the setup phase.

Consequently, the three stages of QA encoding can be certified separately using sumcheck. However, the direct construction requires the prover to commit to the intermediate vectors produced by these stages such as $\mathbf{v}', \mathbf{u}'_1, \dots, \mathbf{u}'_c$, which incurs substantial costs. We show two specific optimizations that eliminate all such intermediate commitments. At a high-level, the final protocol shares a layer-by-layer flavor of GKR[GKR08], reducing output claims to input claims via sumcheck.

Optimization 1: Merging the second and third stages. We observe that each block is generated by the same sequence of operations. They share the same vector $\mathbf{v}'$ and differ only in the public vector $\mathbf{a}'_i$. This common structure allows us to batch all blocks before verifying their encoding relations.

As a result, for every block, we can substitute the pointwise-multiplication relation directly into the final WHT relation and verify for $i \in [c]$,

$$\tilde{u}_i(\gamma) = \sum_{\mathbf{x} \in \{0, 1\}^{n_2}} \tilde{H}_{n_2}(\gamma, \mathbf{x}) 2^{-n_2} \tilde{a}'_i(\mathbf{x}) \tilde{v}'(\mathbf{x}).$$

⁵ We note that zkCNN [LXZ21] shows how to evaluate multilinear extensions of FFT matrices in $O(\log^2 N)$ time. By utilizing the tensor product structure of WHT matrices, our algorithm achieves $O(\log N)$ time.

The intermediate vector $\mathbf{u}'_i$ therefore no longer appears in the verification relation.

The verifier samples a random challenge $\alpha \in \mathbb{F}$ and the parties then verify the single relation

$$\sum_{i=1}^c \alpha^{i-1} \tilde{u}_i(\gamma) = \sum_{i=1}^c \alpha^{i-1} \sum_{\mathbf{x} \in \{0,1\}^{n_2}} \tilde{H}_{n_2}(\gamma, \mathbf{x}) 2^{-n_2} \tilde{a}'_i(\mathbf{x}) \tilde{v}'(\mathbf{x})$$

using one sumcheck. Let $\mathbf{r} \in \mathbb{F}^{n_2}$ be the terminal challenge of this sumcheck. The terminal claim is then reduced to

$$T_1 = \sum_{i=1}^c \alpha^{i-1} \tilde{H}_{n_2}(\gamma, \mathbf{r}) 2^{-n_2} \tilde{a}'_i(\mathbf{r}) \tilde{v}'(\mathbf{r}), \quad (1)$$

where T_1 is the claimed value of the last round. The verifier evaluates $\tilde{H}_{n_2}(\gamma, \mathbf{r})$ locally, while the evaluations $\tilde{a}'_i(\mathbf{r})$ are checked against the preprocessing commitments to the public QA parameters.

This optimization provides two benefits. First, substituting the pointwise-multiplication relation into the final WHT relation eliminates the intermediate vectors $\mathbf{u}'_i$ and hence their commitments. Second, it merges the two sumchecks that would otherwise certify the pointwise multiplication and the final WHT separately, saving $n_2 = \log N_2$ rounds. Moreover, the terminal claim contains only one evaluation of the shared vector $\mathbf{v}'$, which is independent of the number of output blocks.

Optimization 2: Reducing the remaining claim. To verify $\tilde{v}'(\mathbf{r})$, we show that committing to $\mathbf{v}'$ is unnecessary. Define $\kappa(\mathbf{r}) = \sum_{i=1}^c \alpha^{i-1} \tilde{H}_{n_2}(\gamma, \mathbf{r}) 2^{-n_2} \tilde{a}'_i(\mathbf{r})$. Recall from the first encoding step that

$$\tilde{v}'(\mathbf{r}) = \sum_{\mathbf{y} \in \{0,1\}^{n_2}} \tilde{H}_{n_2}(\mathbf{r}, \mathbf{y}) \tilde{v}(\mathbf{y}).$$

Substituting this identity into the equation 1 gives

$$T_1 = \sum_{\mathbf{y} \in \{0,1\}^{n_2}} \kappa(\mathbf{r}) \tilde{H}_{n_2}(\mathbf{r}, \mathbf{y}) \tilde{v}(\mathbf{y}).$$

The parties run a second sumcheck on this relation and reduce to verifying the evaluation of $\tilde{H}_{n_2}(\mathbf{r}, \mathbf{s})$ and $\tilde{v}(\mathbf{s})$, where $\mathbf{s} \in \mathbb{F}^{n_2}$ is the terminal challenge.

The verifier evaluates $\tilde{H}_{n_2}(\mathbf{r}, \mathbf{s})$ locally, and $\tilde{v}(\mathbf{s})$ is an evaluation of the committed input block. Consequently, the protocol does not require an commitment to $\mathbf{v}'$.

As a result, QA code switching can be certified using two sequential sumchecks, without invoking a SPARK-style general-purpose circuit argument and without committing to intermediate encoding witnesses. To the best of our knowledge, existing code-switching constructions based on other linear-time encodable codes, such as RAA codes and Spielman-type codes, do not expose an analogous structure.

3.4 The Setup Phase

In this subsection, we present Algorithm 1 for generating the public parameters of Quasar. The algorithm takes as input the security parameter λ and the number of variables n and outputs $(\mathbf{pp}_{\mathcal{P}}, \mathbf{pp}_{\mathcal{V}})$, where $\mathbf{pp}_{\mathcal{P}}$ is given to the prover and $\mathbf{pp}_{\mathcal{V}}$ is given to the verifier. The algorithm consists of the following three parts:

1. generating the public parameter of BaseFold;
2. generating the public parameter of the QA code;
3. committing to the public vectors $2^{-n_2}\mathbf{a}'_2, \dots, 2^{-n_2}\mathbf{a}'_c$ using the BaseFold PCS.

Let ρ be the inverse rate used in BaseFold and k_0 be a positive integer. The setup algorithm first samples a base generator matrix $\mathbf{G}_0 \in \mathbb{F}^{k_0 \times \rho k_0}$. For every $i > 0$, the generator matrix $\mathbf{G}_i \in \mathbb{F}^{k_i \times \rho k_i}$, $k_i := k_0 \cdot 2^i$ is defined inductively as described in Section 2. This process is repeated until the message length reaches N_2 , since the prover needs to commit to the vector of length N_2 via the BaseFold PCS. Equivalently, assuming that N_2/k_0 is a power of two, we set $\ell := \log_2(N_2/k_0)$ and construct $\mathbf{G}_0, \dots, \mathbf{G}_\ell$.

Let $\mathcal{H}$ be a hash function with λ bits of security. During the BaseFold evaluation protocol, the verifier needs to query random entries of the diagonal matrices $\mathbf{T}_0, \dots, \mathbf{T}_{\ell-1}$. Therefore, for each T_i , the setup algorithm builds a Merkle tree whose leaves are the hashes of the diagonal entries of T_i . We denote the corresponding Merkle root by $\mathbf{rt}_{T_i}$.

For the QA code parameters, the algorithm samples the public vectors $2^{-n_2}\mathbf{a}'_2, \dots, 2^{-n_2}\mathbf{a}'_c \in \mathbb{F}^{N_2}$ uniformly at random, and sets $\mathbf{a}'_1 = \mathbf{1} \in \mathbb{F}^{N_2}$. Since $\mathbf{a}'_2, \dots, \mathbf{a}'_c$ are random, sampling the scaled vectors directly is equivalent to sampling $\mathbf{a}'_i$ uniformly at random and then scaling them. The verifier later needs their multilinear evaluations at random points generated by the QA-encoding checks. Computing these evaluations directly from the full vectors would cost $O(N_2)$ verifier time, which would destroy succinctness. Therefore, their BaseFold commitments can be generated once during the setup phase. Moreover, $\mathbf{a}'_1 = \mathbf{1}$ is explicit, so the verifier can locally evaluate the scaled vector $2^{-n_2}\mathbf{a}'_1$ without a BaseFold commitment.

3.5 The Commit Phase

In this subsection, we give Algorithm 2 for the commitment procedures of Quasar.

The commit algorithm takes as input the public parameters $\mathbf{pp}_{\mathcal{P}}$ and an n -variable multilinear polynomial f , and outputs a commitment $\mathbf{c}$ to f . The algorithm first arranges its evaluations over the Boolean hypercube $\{0, 1\}^n$ as a matrix $\mathbf{M} \in \mathbb{F}^{N_1 \times N_2}$, where $N = 2^n = N_1 N_2$, $N_1 = 2^{n_1}$, $N_2 = 2^{n_2}$. For $\mathbf{i} \in \{0, 1\}^{n_1}$ and $\mathbf{j} \in \{0, 1\}^{n_2}$, the corresponding entry of $\mathbf{M}$ is defined as $M_{\mathbf{i}, \mathbf{j}} = f(\mathbf{i}, \mathbf{j})$. Then the algorithm encodes each row of $\mathbf{M}$ using QA.Enc and obtains a matrix $\mathbf{C} \in \mathbb{F}^{N_1 \times c \cdot N_2}$. Finally, it builds a Merkle tree over the columns of $\mathbf{C}$ and outputs the Merkle root as the commitment to f .

Algorithm 1 Quasar.Gen

Input: Security parameter λ and number of variables n .

Parameter: QA code index c ; parameters $N = 2^n$, $N_1 = O(\log N)$, $N_2 = N/N_1$, $N_1 = 2^{n_1}$, $N_2 = 2^{n_2}$; BaseFold inverse rate ρ ; base message length k_0 .

- 1: Sample a random base generator matrix $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}^{k_0 \times \rho k_0}$.
 - 2: **for** $i = 1$ to $\ell := \log_2(N_2/k_0)$ **do**
 - 3: Set $k_i = k_0 \cdot 2^i$. Sample $\text{diag}(\mathbf{T}_{i-1}) \xleftarrow{\$} (\mathbb{F}^\times)^{\rho k_{i-1}}$.
 - 4: Compute
$$\mathbf{G}_i = \begin{pmatrix} \mathbf{G}_{i-1} & \mathbf{G}_{i-1} \\ \mathbf{G}_{i-1} \cdot \mathbf{T}_{i-1} & \mathbf{G}_{i-1} \cdot -\mathbf{T}_{i-1} \end{pmatrix}.$$
 - 5: Build a Merkle tree $\text{mt}_{T_{i-1}}$ over the diagonal entries of $\mathbf{T}_{i-1}$, and denote its root by $\text{rt}_{T_{i-1}}$.
 - 6: **end for**
 - 7: Sample $c-1$ random vectors $2^{-n_2} \mathbf{a}'_2, \dots, 2^{-n_2} \mathbf{a}'_c \xleftarrow{\$} \mathbb{F}^{N_2}$ and set $\mathbf{a}'_1 = \mathbf{1} \in \mathbb{F}^{N_2}$.
 - 8: Commit to the public vectors $2^{-n_2} \mathbf{a}'_2, \dots, 2^{-n_2} \mathbf{a}'_c \in \mathbb{F}^{N_2}$ using BaseFold PCS denoted by $\mathbf{c}_{a_2}, \dots, \mathbf{c}_{a_c}$.
 - 9: Set $\text{pp}_{\mathcal{P}} = (\mathcal{H}, \mathbf{G}_0, \dots, \mathbf{G}_\ell, \mathbf{T}_0, \dots, \mathbf{T}_{\ell-1}, 2^{-n_2} \mathbf{a}'_2, \dots, 2^{-n_2} \mathbf{a}'_c)$.
 - 10: Set $\text{pp}_{\mathcal{V}} = (\mathcal{H}, \text{rt}_{T_0}, \dots, \text{rt}_{T_{\ell-1}}, \mathbf{c}_{a_2}, \dots, \mathbf{c}_{a_c})$.
 - 11: **return** $\text{pp} = (\text{pp}_{\mathcal{P}}, \text{pp}_{\mathcal{V}})$.
-

Quasar.Open is used only for defining binding and knowledge soundness. It is not invoked in the succinct evaluation protocol. Informally, **Quasar.Open**($\text{pp}, \mathbf{c}, f, \mathbf{h}$) accepts if the hint $\mathbf{h}$ certifies that the Merkle-committed column oracle is within column distance $(\delta_{\text{QA}}/3)cN_2$ from the row-wise QA encoding of f . We give the precise description in Appendix B.

Algorithm 2 Quasar.Commit

Input: prover public parameters $\text{pp}_{\mathcal{P}}$, an n -variable multilinear polynomial f .

Output: Commitment $\mathbf{c}$ and private opening hint $\mathbf{h}$.

- 1: Set the matrix $\mathbf{M} \in \mathbb{F}^{N_1 \times N_2}$ by arranging f 's evaluations over the Boolean hypercube $\{0, 1\}^n$ in row-major fashion.
- 2: **for** $i \in \{0, 1\}^{n_1}$ **do**
- 3: Compute the QA codeword of the i -th row:

$$\mathbf{C}(i, \cdot) := \text{QA.Enc}(\mathbf{M}(i, \cdot)).$$

- 4: **end for**
 - 5: Build a Merkle tree $\mathbf{T}$ over the columns of $\mathbf{C}$ and let $\mathbf{c}$ be its root.
 - 6: Let $\mathbf{h} := (\mathbf{M}, \mathbf{C}, \mathbf{T})$.
 - 7: **return** $(\mathbf{c}, \mathbf{h})$.
-

3.6 The Evaluation Phase

In this subsection, we present Algorithm 3 for the evaluation phase of Quasar. The algorithm takes as input the public parameters $\mathbf{pp}$, a commitment $\mathbf{c}$ to an n -variate multilinear polynomial f , an evaluation point $\mathbf{z} \in \mathbb{F}^n$, and a claimed value $y \in \mathbb{F}$. The prover additionally knows f as a witness. The evaluation phase consists of two parts: a proximity test and an evaluation check. The proximity test proves that the rows of the committed matrix $\mathbf{C}$ are close to QA codewords, while the evaluation check proves that $f(\mathbf{z}) = y$. These two parts have the same structure and are executed in parallel.

Algorithm 3 Quasar.Eval

Input: Public parameters $\mathbf{pp}$, a commitment $\mathbf{c}$, $\mathbf{z} \in \mathbb{F}^n, y \in \mathbb{F}$. $\mathcal{P}$ also has an n -variate multilinear polynomial f .

Output: A bit $b \in \{0, 1\}$.

- 1: $\mathcal{V}$ samples a random $\mathbf{r} \in \mathbb{F}^{N_1}$ and sends it to $\mathcal{P}$.
- 2: Define $\mathbf{z}_L = \otimes_{i=1}^{n_1} (1 - z_i, z_i) \in \mathbb{F}^{N_1}, \mathbf{z}_R = \otimes_{i=n_1+1}^n (1 - z_i, z_i) \in \mathbb{F}^{N_2}$. $\mathcal{P}$ computes $\mathbf{v} = \mathbf{r}^\top \cdot \mathbf{M}, \mathbf{p} = \mathbf{z}_L^\top \cdot \mathbf{M}$ and $\mathbf{u} = \text{QA.Enc}(\mathbf{v}), \mathbf{q} = \text{QA.Enc}(\mathbf{p})$. Denote the intermediate vector by $\mathbf{v}' = \text{WHT}(\mathbf{v}), \mathbf{u}'_i = 2^{-n_2} \mathbf{a}'_i \star \mathbf{v}'$ and $\mathbf{p}' = \text{WHT}(\mathbf{p}), \mathbf{q}'_i = 2^{-n_2} \mathbf{a}'_i \star \mathbf{p}'$ for each $i \in [c]$.
- 3: $\mathcal{P}$ partitions $\mathbf{u}, \mathbf{q}$ into c blocks $\mathbf{u}_1, \dots, \mathbf{u}_c, \mathbf{q}_1, \dots, \mathbf{q}_c$ respectively and commits to them using the BaseFold PCS denoted by $\mathbf{c}_{\mathbf{u}_1}, \dots, \mathbf{c}_{\mathbf{u}_c}, \mathbf{c}_{\mathbf{q}_1}, \dots, \mathbf{c}_{\mathbf{q}_c}$. Then $\mathcal{P}$ sends the commitments to $\mathcal{V}$.
- 4: $\mathcal{V}$ sends a random set $I \subset \{0, \dots, cN_2 - 1\}$ of size $t = \Theta(\lambda)$ to $\mathcal{P}$. Let $I = \{i_0, \dots, i_{t-1}\}$.
- 5: For every $i \in I$, $\mathcal{P}$ opens the columns $\mathbf{C}(\cdot, i)$ and $\mathcal{V}$ verifies the Merkle path proofs.
- 6: $\mathcal{V}$ sends a random value $\tau \in \mathbb{F}$ to $\mathcal{P}$. The parties define the selector polynomial $\tilde{h} : \mathbb{F}^{\log cN_2} \rightarrow \mathbb{F}$ by $\tilde{h}(\mathbf{x}) = \tau^j$ if $\mathbf{x} = \langle i_j \rangle$ for $j = 0, \dots, t-1$, and $\tilde{h}(\mathbf{x}) = 0$ otherwise.
- 7: The parties run a sumcheck protocol in parallel to check that $\sum_{\mathbf{x} \in \{0,1\}^{\log cN_2}} \tilde{h}(\mathbf{x}) \tilde{u}(\mathbf{x}) = \sum_{j=0}^{t-1} \tau^j \langle \mathbf{r}, \mathbf{C}(\cdot, i_j) \rangle$ and $\sum_{\mathbf{x} \in \{0,1\}^{\log cN_2}} \tilde{h}(\mathbf{x}) \tilde{q}(\mathbf{x}) = \sum_{j=0}^{t-1} \tau^j \langle \mathbf{z}_L, \mathbf{C}(\cdot, i_j) \rangle$. Finally, $\mathcal{V}$ needs to evaluate $\tilde{u}_1, \dots, \tilde{u}_c, \tilde{q}_1, \dots, \tilde{q}_c$ at a random point $\mathbf{e}_1$, whereas $\mathcal{V}$ evaluates $\tilde{h}$ on its own.
- 8: $\mathcal{P}$ sends the evaluations of $\tilde{u}_1, \dots, \tilde{u}_c, \tilde{q}_1, \dots, \tilde{q}_c$ at $\mathbf{e}_1$ to $\mathcal{V}$. $\mathcal{V}$ finishes the last round of the sumcheck protocol. If the check fails, it outputs 0 and halts.
- 9: $\mathcal{V}$ samples random points $\gamma \leftarrow \mathbb{F}^{n_2}$ and a batching challenge $\alpha \leftarrow \mathbb{F}$.
- 10: $\mathcal{P}$ sends the claimed evaluations $\tilde{u}_i(\gamma), \tilde{q}_i(\gamma)$ to $\mathcal{V}$ for $i \in [c]$.
- 11: The parties run two sumchecks in parallel to prove that

$$\sum_{i=2}^c \alpha^{i-2} \tilde{u}_i(\gamma) = \sum_{i=2}^c \alpha^{i-2} \sum_{\mathbf{x} \in \{0,1\}^{n_2}} \tilde{H}_{n_2}(\gamma, \mathbf{x}) 2^{-n_2} \tilde{a}'_i(\mathbf{x}) \tilde{v}'(\mathbf{x})$$

$$\sum_{i=2}^c \alpha^{i-2} \tilde{q}_i(\gamma) = \sum_{i=2}^c \alpha^{i-2} \sum_{\mathbf{x} \in \{0,1\}^{n_2}} \tilde{H}_{n_2}(\gamma, \mathbf{x}) 2^{-n_2} \tilde{a}'_i(\mathbf{x}) \tilde{p}'(\mathbf{x})$$

- 12: Let T_1, T_2 be the claimed value of the last round and $\mathbf{e}_2 \in \mathbb{F}^{n_2}$ is the final challenge. The parties run two sumchecks in parallel to prove that

$$T_1 = \sum_{\mathbf{y} \in \{0,1\}^{n_2}} \sum_{i=2}^c \alpha^{i-2} \tilde{H}_{n_2}(\gamma, \mathbf{e}_2) 2^{-n_2} \tilde{a}'_i(\mathbf{e}_2) \tilde{H}_{n_2}(\mathbf{e}_2, \mathbf{y}) \tilde{v}(\mathbf{y}),$$

$$T_2 = \sum_{\mathbf{y} \in \{0,1\}^{n_2}} \sum_{i=2}^c \alpha^{i-2} \tilde{H}_{n_2}(\gamma, \mathbf{e}_2) 2^{-n_2} \tilde{a}'_i(\mathbf{e}_2) \tilde{H}_{n_2}(\mathbf{e}_2, \mathbf{y}) \tilde{p}(\mathbf{y}).$$

Finally, $\mathcal{V}$ needs to evaluate $\tilde{v}, \tilde{p}$ at a random point $\mathbf{e}_3$ and $\tilde{a}'_i$ at $\mathbf{e}_2$ for $i = 2, \dots, c$. $\mathcal{P}$ sends these values and $\mathcal{V}$ checks the final claim of the batched sumcheck. If the check fails, $\mathcal{V}$ outputs 0 and halts.

- 13: The parties run one batched BaseFold evaluation protocol to check all claimed evaluations from Steps 8, 10, and 12, together with the final evaluation claim $\tilde{q}_1(\mathbf{z}_R) = y$. The verifier outputs 1 iff the batched BaseFold verifier accepts.

In conclusion, we formally give the theorem below.

Theorem 3.1. *Let $\mathbb{F}$ be a finite field of odd characteristic, let $N = 2^n = N_1 N_2$, and let $\mathcal{C}_{\text{QA}} \leftarrow \text{QA.Gen}$ be a systematic QA code of inverse rate c such that $\Pr[\delta(\mathcal{C}_{\text{QA}}) < \delta_{\text{QA}}] \leq \varepsilon_{\text{dist}}$. Assume that the underlying BaseFold PCS has knowledge error ε_{BF} and that the vector commitment used to authenticate the IOP oracles satisfies the security requirement of the IBCS compiler [CDGS23], with error ε_{VC} . Then $(\text{Quasar.Gen}, \text{Quasar.Commit}, \text{Quasar.Open}, \text{Quasar.Eval})$ is an extractable succinct interactive polynomial commitment scheme for multilinear polynomials, with knowledge error $\varepsilon_{\text{Quasar}} \leq \varepsilon_{\text{dist}} + \varepsilon_{\text{IOP}} + \varepsilon_{\text{VC}}$, where ε_{IOP} is the conditional error established in Appendix B when $\delta(\mathcal{C}_{\text{QA}}) \geq \delta_{\text{QA}}$.*

For constant c and $N_1 = \Theta(\log N)$, the commitment algorithm performs $O(N \log N)$ field additions and $O(N)$ field multiplications. The evaluation prover runs in $O(N)$ time, while verifier time and proof size are both $O(\lambda \log^2 N)$.

The efficiency analysis and the proof of Theorem 3.1 are deferred to Appendix A, B.

4 Implementation and Evaluation

In this section, we describe our implementation of **Quasar** and evaluate its concrete performance. We first present a GPU implementation of the QA encoder and explain why the structure of QA encoding maps naturally to GPUs. We then integrate this encoder into the commitment phase of **Quasar** through a hybrid GPU–CPU Merkle pipeline, and compare the resulting implementation with BaseFold, Brakedown, and the Brakedown-style PCS based on QA codes without code switching, which we denote by QAPCS.

All experiments are conducted on a server equipped with an AMD EPYC 9754 128-Core Processor, 512 GB of RAM, and an NVIDIA GeForce RTX 5090 GPU,

running Ubuntu 22.04.5 LTS. Our implementation is built on top of the public `plonkish_basefold` codebase⁶, which contains proof-of-concept implementations of BaseFold and Brakedown. We use this codebase as the basis for the BaseFold and Brakedown-style baselines and implement Brakingbase, Quasar and QAPCS within the same framework. Our implementation of Quasar is publicly available at <https://anonymous.4open.science/r/Quasar-33F6>.

4.1 GPU Acceleration of QA Encoding

QA encoding consists almost of data-parallel operations and is therefore particularly suitable for GPU acceleration. We view this as an additional advantage of QA-based schemes, whereas the direct and practical GPU efficiency of other fast-code encoders remains less clear.

Mapping QA encoding to the GPU. Recall that the systematic rate- $1/c$ QA encoder maps a message $\mathbf{v} \in \mathbb{F}^N$ to

$$\text{Enc}_{\text{QA}}(\mathbf{v}) = (\mathbf{v}, \mathbf{z}_1, \dots, \mathbf{z}_{c-1}),$$

where, for every $i \in [c-1]$,

$$\mathbf{v}' = \text{WHT}(\mathbf{v}), \quad \mathbf{u}_i = \mathbf{v}' \odot \mathbf{a}'_i, \quad \mathbf{z}_i = \text{WHT}(\mathbf{u}_i).$$

Thus, QA encoding consists of two WHTs and coordinate-wise multiplication with public vectors. This structure maps naturally to a GPU. Each WHT layer contains a large number of independent structured butterfly operations, whereas the middle layer consists of independent field multiplications. Consequently, the encoder exposes abundant parallelism and regular memory-access patterns, both of which are well suited to GPU execution. Moreover, the public coefficient vectors $\mathbf{a}'_i$ can remain resident on the device and be reused across encoded rows.

An important advantage is that this GPU mapping directly follows the native QA encoding algorithm. Although QA encoding can be viewed as multiplication by a dense generator matrix, our implementation never materializes that matrix. Instead, it evaluates the WHT-based encoder directly, using only regular butterfly operations and pointwise multiplications.

This direct mapping contrasts with TensorZKP [LZG⁺26], which targets GPU Tensor Cores specialized for matrix multiplication. To use these units, TensorZKP reformulates its sumcheck and Spielman-code encoding as matrix-multiplication tasks. Our approach is different: we exploit the regular WHT structure in QA encoding on conventional CUDA cores. Thus, QA encoding does not require a reduction to matrix multiplication, thereby avoiding the movement overheads.

Encoding experiment. We evaluate a single rate- $1/2$ QA codeword with message length 2^k for $k \in \{12, \dots, 25\}$. We compare three implementations: the original serial CPU encoder, a 32-thread CPU encoder that parallelizes the WHT butterflies and pointwise products within one codeword, and the GPU encoder.

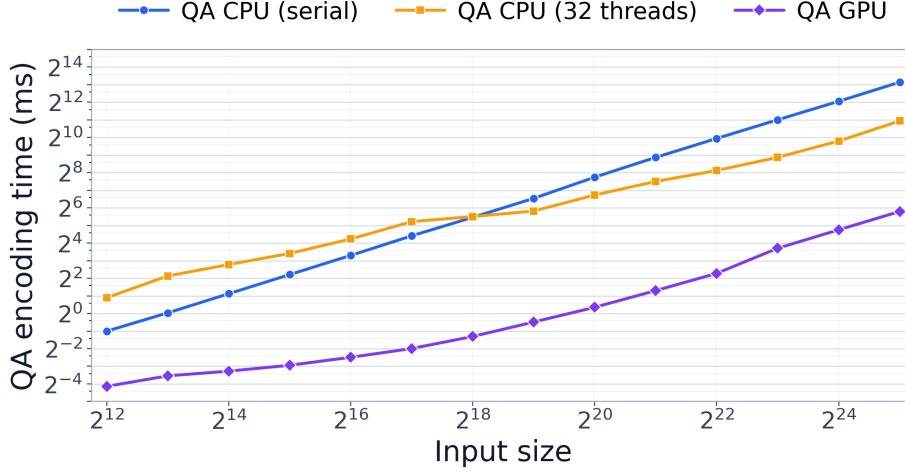


Fig. 1: QA encoding time over Mersenne127 at rate 1/2.

For each size, we perform five warm-up runs followed by ten measured runs and report the mean.

Figure 1 shows that parallelizing a single codeword on the CPU is beneficial only once the message reaches 2^{19} elements. Below this point, task scheduling and the barrier between consecutive WHT stages dominate the available parallelism. At larger sizes, the parallel CPU implementation becomes increasingly effective: at 2^{25} elements, it reduces the encoding time from 9035.19 ms to 1962.29 ms, a $4.60\times$ speedup over the serial implementation.

The GPU implementation is faster throughout the tested range. Across message lengths from 2^{12} to 2^{25} , the GPU encoder achieves an $8.8\times$ – $203.1\times$ speedup over the serial CPU implementation and a $32.9\times$ – $148.3\times$ speedup over the 32-thread CPU implementation. At 2^{25} elements, the GPU takes 55.53 ms, yielding a $162.7\times$ speedup over the serial CPU implementation and a $35.3\times$ speedup over the 32-thread implementation.

4.2 Quasar Performance

GPU-accelerated commitment. The commitment phase arranges the evaluations of the multilinear polynomial into a matrix, applies the QA encoder to each row, and constructs a Merkle tree over the encoded columns. Using the standalone GPU encoder alone would still require transferring the entire encoded matrix back to the CPU for Merkle hashing, which would substantially diminish the benefit of acceleration.

We therefore adopt a hybrid GPU–CPU commitment pipeline. The GPU performs the data-parallel tasks: QA encoding, hashing the encoded columns and

⁶ https://github.com/hadasz/plonkish_basefold

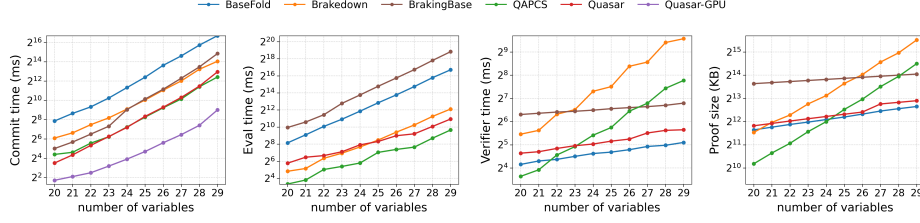


Fig. 2: Performance comparison at rate 1/2. The additional **Quasar-GPU** curve appears in the commitment panel.

gathering the selected columns. Only the resulting column digests are returned for Merkle-tree construction, and only the queried columns are transferred to the CPU when openings are needed. The CPU constructs the upper Merkle levels and executes the subsequent proof procedures. This model preserves the original commitment and proof interfaces: the commitment root, authentication paths, proof, and verifier algorithm remain unchanged. Thus, the hybrid design captures the main benefit of GPU acceleration without requiring a full migration of the prover to the GPU.

Experimental setup. All CPU implementations use 32 threads. We instantiate every scheme over the 127-bit Mersenne prime field, set the code rate to 1/2. For Quasar, we target 100-bit security. The concrete parameters are reported in Table 4. We evaluate multilinear polynomials of size 2^k for $k \in \{20, \dots, 29\}$. For each setting, we run the protocol ten times and report the average. We measure commitment time, evaluation time, verifier time, and proof size.

GPU commitment results. As shown in the first panel of Figure 2, GPU acceleration reduces the **Quasar** commitment time across the entire tested range. The GPU commitment takes 3.27 ms at $N = 2^{20}$ and 512.80 ms at $N = 2^{29}$. Relative to the 32-thread CPU implementation of **Quasar**, this corresponds to a $3.4\times$ – $16.7\times$ speedup. The increasing advantage at large sizes confirms that keeping the encoded matrix on the device and transferring only Merkle digests is essential.

Comparison with BaseFold. Compared with BaseFold, **Quasar** substantially reduces prover-side costs across the entire tested range. For $N = 2^{20}$ – 2^{29} , **Quasar** is $13.6\times$ – $20.6\times$ faster in commitment and $5.2\times$ – $63.8\times$ faster in evaluation. The verifier time and proof size remain close. Over $N = 2^{20}$ – 2^{29} , **Quasar** is only $1.3\times$ – $1.6\times$ slower in verification. For $N = 2^{20}$ – 2^{26} , their proof sizes differ by at most 10%. The hybrid GPU-CPU implementation further improves the commitment phase without changing these succinct verifier and proof-size properties.

Comparison with Brakedown. Compared with Brakedown, **Quasar** improves all four measured metrics over their respective advantage ranges. For $N = 2^{20}$ – 2^{29} , **Quasar** is $2.2\times$ – $6.0\times$ faster in commitment and $1.8\times$ – $14.8\times$ faster in verification. For $N = 2^{21}$ – 2^{29} , it produces $1.1\times$ – $6.3\times$ smaller proofs. These verifier-time

and proof-size advantages become increasingly pronounced at larger input sizes, reflecting the polylogarithmic verifier and proof complexity of **Quasar**, in contrast to the square-root costs of **Brakedown**. From $N = 2^{25}$ to 2^{29} , **Quasar** is also $1.1\times$ – $2.3\times$ faster in evaluation.

Comparison with QAPCS. The comparison with QAPCS demonstrates the cost and benefit of the code-switching layer, since both schemes use the same QA-code commitment front end. At small input sizes, QAPCS has lower evaluation costs and smaller proofs because of its simpler protocol structure. As the input size increases, however, the benefits of code switching become increasingly pronounced. For $N = 2^{24}$ – 2^{29} , **Quasar** is $1.3\times$ – $4.3\times$ faster in verification. For $N = 2^{25}$ – 2^{29} , it produces $1.2\times$ – $2.9\times$ smaller proofs. At the largest tested size $N = 2^{29}$, QAPCS requires 218 ms for verification and produces a 23120 KB proof, whereas **Quasar** requires approximately 51 ms and produces an 8049 KB proof.

Comparison with BrakingBase. Although **BrakingBase** has linear prover complexity, its concrete prover costs remain higher in our experiments. Across sizes from 2^{20} to 2^{29} , **Quasar** is $2.1\times$ – $4.0\times$ faster in commitment and $17.4\times$ – $237.6\times$ faster in evaluation. **Quasar** also achieves a $2.1\times$ – $3.2\times$ faster verifier and produces $2.2\times$ – $3.5\times$ smaller proofs.

4.3 Matrix-multiplication verification

Following DeepFold [GLH⁺25], we evaluate the four PCS schemes in an application-level benchmark for verifying a matrix multiplication relation $\mathbf{AB} = \mathbf{C}$, where $\mathbf{A} \in \mathbb{F}^{32 \times n}$, $\mathbf{B} \in \mathbb{F}^{n \times p}$, and $\mathbf{C} \in \mathbb{F}^{32 \times p}$. The sumcheck protocol reduces the matrix-multiplication identity to a small number of multilinear evaluation claims, which are then discharged using the corresponding PCS. We report $k \in \{25, 27, 29\}$, where $k = \log_2(np)$ is the logarithm of the number of entries in the larger input matrix $\mathbf{B}$. Specifically, the corresponding (n, p) values are $(2^{12}, 2^{13})$, $(2^{13}, 2^{14})$, and $(2^{14}, 2^{15})$, respectively.

Table 3 reports the results. The prover time includes the input commitments, application preparation, sumcheck proving, and PCS openings. The verifier time includes both sumcheck and PCS verification, while the reported proof size includes the input commitments. We omit **BrakingBase** from the matrix-multiplication benchmark because its implementation ran out of memory at the tested scales, potentially due to the memory overhead of its SPARK-based verification.

Across the three matrix sizes, **Quasar** is $12.2\times$ – $14.9\times$ faster than **BaseFold** in total prover time, with comparable verifier time and proof size. Compared with **Brakedown**, its prover time is within 8% at $k = 25, 27$ and $1.1\times$ faster at $k = 29$, while achieving $3.9\times$ – $12.6\times$ faster verification and $1.9\times$ – $4.9\times$ smaller proofs. QAPCS remains $3.1\times$ – $3.9\times$ faster in proving, but **Quasar** scales better on the verifier side: at $k = 29$, it verifies in 71 ms versus 221 ms and produces a $2.2\times$ smaller proof.

Table 3: Matrix-multiplication verification performance.

k	System	Prover (s)	Verifier (s)	Proof (MB)
25	BaseFold	14.98	0.066	10.02
	Brakedown	1.15	0.224	16.67
	QAPCS	0.34	0.064	6.90
	Quasar	1.21	0.057	8.59
27	BaseFold	60.47	0.072	11.37
	Brakedown	4.61	0.436	29.53
	QAPCS	1.27	0.114	12.86
	Quasar	4.97	0.066	9.80
29	BaseFold	254.24	0.081	12.81
	Brakedown	19.56	0.896	54.63
	QAPCS	5.43	0.221	24.57
	Quasar	17.05	0.071	11.09

References

- ACFY24. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: reed-solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, Lecture Notes in Computer Science, pages 380–413. Springer, 2024. [2](#), [3](#), [5](#)
- ACFY25. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: reed-solomon proximity testing with super-fast verification. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*, Lecture Notes in Computer Science, pages 214–243. Springer, 2025. [2](#), [3](#), [5](#)
- AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. Liger: Lightweight sublinear arguments without a trusted setup. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017. [2](#), [3](#), [5](#), [33](#)
- BBB⁺18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018. [2](#)
- BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, July 9-13, 2018*, LIPIcs, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. [2](#), [3](#), [5](#)

- BCF⁺25. Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast snarks from interleaved RAA codes. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*, Lecture Notes in Computer Science, pages 123–152. Springer, 2025. [2](#), [3](#), [4](#), [5](#), [6](#)
- BCG20. Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, Lecture Notes in Computer Science, pages 19–46. Springer, 2020. [2](#)
- BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, Lecture Notes in Computer Science, pages 31–60, 2016. [32](#)
- BFH⁺20. Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkatasubramanian, Tiancheng Xie, and Yupeng Zhang. Liger++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020. [2](#)
- BFK⁺24. Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, Lecture Notes in Computer Science, pages 276–307. Springer, 2024. [3](#), [4](#), [6](#)
- BFMM26. Anubhav Baweja, Giacomo Fenzi, Pratyush Mishra, and Tushar Mopuri. Field-agnostic SNARKs with small proofs via encode-repeat-accumulate (ERA) codes. Cryptology ePrint Archive, Paper 2026/864, 2026. [2](#), [3](#), [7](#)
- BMMS25. Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, and Matan Shtepele. Query-optimal iopps for linear-time encodable codes. *IACR Cryptol. ePrint Arch.*, 2025:1588, 2025. [3](#), [4](#)
- 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 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, Lecture Notes in Computer Science, pages 499–530. Springer, 2023. [7](#), [16](#)
- CDGS23. Alessandro Chiesa, Marcel Dall'Agnol, Ziyi Guan, and Nicholas Spooner. On the security of succinct interactive arguments from vector commitments. Cryptology ePrint Archive, Paper 2023/1737, 2023. [21](#), [39](#)
- CYY26. Mingshu Cong, Tsz Hon Yuen, and Siu-Ming Yiu. SwitchFold: Code-agnostic succinct polynomial commitments via recursive code switching. Cryptology ePrint Archive, Paper 2026/1489, 2026. [7](#), [41](#)
- GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor,

- Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122. ACM, 2008. [6](#), [16](#)
- GLH⁺25. Yanpei Guo, Xuanming Liu, Kexi Huang, Wenjie Qu, Tianyang Tao, and Jiaheng Zhang. Deepfold: Efficient multilinear polynomial commitment from reed-solomon code and its application to zero-knowledge proofs. In Lujo Bauer and Giancarlo Pellegrino, editors, *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 3497–3516. USENIX Association, 2025. [2](#), [25](#)
- GLS⁺23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, Lecture Notes in Computer Science, pages 193–226. Springer, 2023. [2](#), [3](#), [4](#), [5](#)
- GNR26. Kobi Gurkan, Andrija Novakovic, and Ron D. Rothblum. Bolt: Faster SNARKs from sketched codes. Cryptology ePrint Archive, Paper 2026/310, 2026. [2](#), [3](#), [7](#)
- GPS25. Chaya Ganesh, Sikhar Patranabis, and Nitin Singh. Samaritan: Linear-time prover snark from new multilinear polynomial commitments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 456–489. Springer, 2025. [2](#)
- Hab23. Ulrich Haböck. Brakedown’s expander code. *IACR Cryptol. ePrint Arch.*, 2023:769, 2023. [3](#)
- KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, Lecture Notes in Computer Science, pages 177–194. Springer, 2010. [2](#)
- Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, Lecture Notes in Computer Science, pages 1–34. Springer, 2021. [2](#)
- LFKN90. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 2–10. IEEE Computer Society, 1990. [8](#)
- LLX⁺26a. Zhe Li, Hongqing Liu, Chaoping Xing, Yizhou Yao, and Chen Yuan. Faster pseudorandom correlation generators via Walsh–Hadamard transform. In *Advances in Cryptology - CRYPTO 2026*, 2026. To appear. [3](#), [15](#)
- LLX⁺26b. Zhe Li, Hongqing Liu, Chaoping Xing, Yizhou Yao, and Chen Yuan. More efficient SNARKs via quasi-abelian codes: Faster, smaller, and field-agnostic. Cryptology ePrint Archive, Paper 2026/939, 2026. [3](#), [4](#), [5](#), [11](#), [15](#)
- LXZ21. Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkcn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS ’21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual*

- Event, Republic of Korea, November 15 - 19, 2021, pages 2968–2985. ACM, 2021. [16](#)
- LZG⁺26. Tao Lu, Jipeng Zhang, Yanpei Guo, Xuanming Liu, Wenjie Qu, Zonghui Wang, Wenzhi Chen, and Jiaheng Zhang. TensorZKP: Repurposing GPU tensor cores for high-performance zero-knowledge proofs. In *Proceedings of the 35th USENIX Security Symposium (USENIX Security '26)*, Baltimore, MD, USA, August 2026. USENIX Association. [22](#)
- NST25. Vineet Nair, Ashish Sharma, and Bhargav Thankey. Brakingbase - A linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part V*, Lecture Notes in Computer Science, pages 427–455. Springer, 2025. [2](#), [3](#), [4](#), [5](#), [7](#), [11](#), [34](#)
- PST13. Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography Conference*, pages 222–242. Springer, 2013. [2](#)
- QGZ26. Wenjie Qu, Yanpei Guo, and Jiaheng Zhang. Lightning, field-agnostic super-efficient polynomial commitment scheme. Cryptology ePrint Archive, Paper 2026/258, 2026. [2](#), [3](#), [7](#)
- RZR24. Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Journal of the ACM*, 71(3):1–42, 2024. [3](#)
- Set20. Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, Lecture Notes in Computer Science, pages 704–737. Springer, 2020. [5](#), [6](#), [16](#)
- Spi95. Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 388–397. ACM, 1995. [3](#)
- Tha13. Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, Lecture Notes in Computer Science, pages 71–89. Springer, 2013. [8](#)
- Tha22. Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. [33](#)
- WTS⁺18. Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018. [2](#)
- XZS22. Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, Lecture Notes in Computer Science, pages 299–328. Springer, 2022. [2](#), [3](#), [5](#), [6](#)

- ZCF24. Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, Lecture Notes in Computer Science, pages 138–169. Springer, 2024. [2](#), [3](#), [5](#), [10](#), [33](#)
- ZXZS20. Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 859–876. IEEE, 2020. [6](#)

Supplementary Material

A Efficiency

In this section, we analyze the verifier time, prover time, and proof size of **Quasar**.

Recall that $N = 2^n$, $N_1 = \Theta(\log N)$, and $N_2 = N/N_1$. The verifier samples $t = \Theta(\lambda)$ columns in the proximity test. Throughout this section, we treat the QA code index c and the BaseFold inverse rate ρ as fixed constants.

Verifier time. The verifier running time is dominated by the following components:

1. verifying the Merkle opening proofs for the sampled columns $\mathbf{C}(\cdot, j)$ for all $j \in I$;
2. executing the two selector-sumchecks in Steps 7–8;
3. executing the sumchecks in Steps 11–12;
4. executing the final batched BaseFold evaluation protocol in Step 13.

For Item 1, each opened column has length $N_1 = \Theta(\log N)$ and each Merkle path has length $O(\log(cN_2)) = O(\log N)$. Since the verifier opens $t = \Theta(\lambda)$ columns, this costs $O(\lambda \log N)$ field and hash operations.

For Item 2, the verifier computes the right-hand sides of the two selector-sumcheck claims using the opened columns. This costs $O(\lambda N_1) = O(\lambda \log N)$ field operations. The verifier also evaluates the sparse selector polynomial $\tilde{h}$, which costs $O(\lambda \log N)$. Thus, the total verifier time for the selector-sumchecks is $O(\lambda \log N)$.

For Item 3, the verifier performs $O(\log N_2) = O(\log N)$ field operations. It also locally evaluates the tensor-structured WHT polynomial $\tilde{H}_{n_2}$ and the equality polynomial, each in $O(n_2) = O(\log N)$ time.

For Item 4, the number of batched claims is $O(c)$, and then the verifier time is therefore $O(c\lambda \log^2 N_2) = O(\lambda \log^2 N)$.

Combining the above terms, the verifier running time is $O(\lambda \log^2 N)$.

Prover time. We include both the commitment phase and the evaluation phase in the prover time.

In the commitment phase, the prover first arranges the evaluations of f over $\{0, 1\}^n$ into a matrix $M \in \mathbb{F}^{N_1 \times N_2}$. It then encodes each row of M using **QA.Enc** and obtains the matrix $\mathbf{C} \in \mathbb{F}^{N_1 \times cN_2}$. For one row of length N_2 , the QA encoding costs $O(cN_2 \log N_2)$ field additions and $O(cN_2)$ field multiplications. Since there are N_1 rows, the total QA encoding cost in the commitment phase is $O(cN_1 N_2 \log N_2) = O(cN \log N_2)$ field additions and $O(cN_1 N_2) = O(cN)$ field multiplications. Building the Merkle tree over the columns of $\mathbf{C}$ requires hashing $O(cN)$ field elements. Thus, for constant c , the commitment phase has quasi-linear additive complexity and linear multiplication complexity. Remark that over the large prime fields targeted by our implementation, field multiplications are substantially more expensive than additions and usually dominate the concrete arithmetic cost.

In the evaluation phase, the prover time is dominated by the following components:

1. computing the two linear combinations $\mathbf{v} = \mathbf{r}^\top \mathbf{M}$ and $\mathbf{p} = \mathbf{z}_L^\top \mathbf{M}$;
2. computing the QA encodings $\mathbf{u} = \text{QA.Enc}(\mathbf{v})$ and $\mathbf{q} = \text{QA.Enc}(\mathbf{p})$;
3. committing to the blocks of $\mathbf{u}, \mathbf{q}$ using BaseFold;
4. running the selector-sumcheck in Steps 7–8;
5. running the batched sumcheck in Steps 11–12;
6. running the final batched BaseFold evaluation protocol in Step 13.

For Item 1, computing $\mathbf{v} = \mathbf{r}^\top \mathbf{M}$ and $\mathbf{p} = \mathbf{z}_L^\top \mathbf{M}$ costs $O(N_1 N_2) = O(N)$ field operations.

For Item 2, the QA encoding costs $O(cN_2 \log N_2)$ field additions and $O(cN_2)$ field multiplications.

For Item 3, the prover commits to $O(c)$ vectors of length N_2 using BaseFold. Thus, the total cost is $O(cN_2 \log N_2) = O(N)$.

For Items 4 and 5, the domain size of two sumchecks are $O(cN_2)$ and $O(N_2)$. Using the standard linear-time sumcheck prover, their total prover time is $O(cN_2) = O(N)$.

For Item 6, the batched BaseFold evaluation prover is invoked on $O(c)$ committed vectors of length N_2 . Its cost is linear in N_2 .

Therefore, the evaluation time is $O(N)$.

Proof size. The proof consists of the following parts:

1. the opened columns $\mathbf{C}(\cdot, j)$ for $j \in I$ and their Merkle authentication paths;
2. the messages of the selector-sumchecks and the QA-encoding sumchecks;
3. the commitments to the blocks of $\mathbf{u}, \mathbf{q}$ using BaseFold;
4. the final batched BaseFold evaluation proof.

The opened columns contribute $O(\lambda N_1) = O(\lambda \log N)$ field elements, and the corresponding Merkle paths contribute $O(\lambda \log N)$ hash values. The sumcheck messages contribute $O(\log N)$ field elements for a constant number of batched sumchecks. The auxiliary commitments contribute $O(c)$ hash values. The dominant term is the final batched BaseFold evaluation proof, whose size is $O(\lambda \log^2 N_2) = O(\lambda \log^2 N)$.

Hence, the total proof size is $O(\lambda \log^2 N)$.

B Deferred Proofs

We prove Theorem 3.1 in the IOP [BCS16] model. In this model, the public input contains oracle access to the committed matrix C , rather than only the Merkle root $\mathbf{c}$.

Recall the notations from Section 3. Let $f(X_1, \dots, X_n) \in \mathbb{F}[X_1, \dots, X_n]$ be multilinear, and let $N = 2^n = N_1 N_2$, where $N_1 = 2^{n_1}$ and $N_2 = 2^{n_2}$. The prover arranges the evaluations of f over $\{0, 1\}^n$ as a matrix $\mathbf{M} \in \mathbb{F}^{N_1 \times N_2}$. For $\mathbf{i} \in \{0, 1\}^{n_1}$ and $\mathbf{j} \in \{0, 1\}^{n_2}$, we write $\mathbf{M}(\mathbf{i}, \mathbf{j}) = f(\mathbf{i}, \mathbf{j})$. The committed

matrix is $\mathbf{C} \in \mathbb{F}^{N_1 \times cN_2}$, where $\mathbf{C}(\mathbf{i}, \cdot) = \text{QA.Enc}(\mathbf{M}(\mathbf{i}, \cdot))$. Let δ_{QA} be the relative distance of the QA code, and write $L := cN_2$ for the QA codeword length.

For an evaluation point $\mathbf{z} = (z_1, \dots, z_n) \in \mathbb{F}^n$, define $\mathbf{z}_L = \otimes_{i=1}^{n_1} (1 - z_i, z_i) \in \mathbb{F}^{N_1}$ and $\mathbf{z}_R = \otimes_{i=n_1+1}^n (1 - z_i, z_i) \in \mathbb{F}^{N_2}$. Then $f(\mathbf{z}) = \langle \mathbf{z}_L^\top \mathbf{M}, \mathbf{z}_R \rangle$. In the evaluation protocol, the prover computes $\mathbf{v} = \mathbf{r}^\top \mathbf{M}$ and $\mathbf{p} = \mathbf{z}_L^\top \mathbf{M}$, and then commits to $\mathbf{u} = \text{QA.Enc}(\mathbf{v})$ and $\mathbf{q} = \text{QA.Enc}(\mathbf{p})$, where $\mathbf{u} = (\mathbf{u}_1 \parallel \dots \parallel \mathbf{u}_c)$ and $\mathbf{q} = (\mathbf{q}_1 \parallel \dots \parallel \mathbf{q}_c)$. Since the QA code is systematic, $\mathbf{u}_1 = \mathbf{v}$ and $\mathbf{q}_1 = \mathbf{p}$.

The opening algorithm. We first make explicit the opening algorithm used in the binding and knowledge soundness.

Given public parameters pp , a commitment $\mathbf{c}$, an n -variate multilinear polynomial f , and an opening hint $\mathbf{h} = (\mathbf{M}, \mathbf{C}, \mathcal{T})$, the predicate $\text{Quasar.Open}(\text{pp}, \mathbf{c}, f, \mathbf{h})$ is defined as follows.

The algorithm first checks that $\mathbf{M}$ is the evaluation table of f over $\{0, 1\}^n$. It then verifies that $\mathcal{T}$ is a Merkle tree whose leaves are the columns of $\mathbf{C}$ and whose root is equal to $\mathbf{c}$. If either check fails, it rejects.

Next, the algorithm computes the row-wise QA encoding $\mathbf{C}_f(\mathbf{i}, \cdot) := \text{QA.Enc}(\mathbf{M}(\mathbf{i}, \cdot))$ for every $\mathbf{i} \in \{0, 1\}^{n_1}$. It accepts if and only if

$$\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}_f) \leq \frac{\delta_{\text{QA}}}{3} cN_2.$$

The radius $\delta_{\text{QA}}/3$ is strictly below half the relative distance of the QA code. Consequently, two distinct multilinear polynomials cannot both be accepted as openings of the same committed column oracle.

Tools used in the proof. We first give some tools used in the proof.

Lemma B.1 (Lemma 12.1 of [Tha22]). *Let X, Y be jointly distributed random variables, and let $E(X, Y)$ be an event such that $\Pr_{X,Y}[E(X, Y)] \geq e$. Let $S := \{x : \Pr_Y[E(X, Y) \mid X = x] \geq e/2\}$. Then $\Pr_X[X \in S] \geq e/2$.*

Lemma B.2 (Lemma 9 of [ZCF24]). *Let $\Phi : \mathcal{M}^* \rightarrow \{0, 1\}$ be any predicate such that, for any $(m_1, \dots, m_i) \in \mathcal{M}^i$ with $\Phi(m_1, \dots, m_i) = 1$, it holds that $\Pr_{m_{i+1} \leftarrow_R \mathcal{M}}[\Phi(m_1, \dots, m_i, m_{i+1}) = 1] \geq 1 - \text{negl}(\lambda)$. Let $N = \text{poly}(\lambda)$. For any $e > 0$, there exists an extractor Ext running in time $T \in O(\lambda/e)$ such that, given oracle access to any algorithm $\mathcal{A}$ satisfying $\Pr_{m \leftarrow_R \mathcal{M}}[\mathcal{A}(m) = 1] \geq e$, the following holds:*

$$\Pr \left[\begin{array}{l} \Phi(m_1, \dots, m_N) = 1 \\ \wedge \mathcal{A}(m_i) = 1 \ \forall i \in [N] \end{array} \middle| (m_1, \dots, m_N) \leftarrow \text{Ext}^{\mathcal{A}} \right] \geq 1 - T \cdot \text{negl}(\lambda).$$

For two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{F}^{N_1 \times L}$, let $\Delta_{\text{col}}(\mathbf{A}, \mathbf{B}) := |\{j \in [L] : \mathbf{A}(\cdot, j) \neq \mathbf{B}(\cdot, j)\}|$. For an interleaved code $\mathcal{D} \subseteq \mathbb{F}^{N_1 \times L}$, let $\Delta_{\text{col}}(\mathbf{A}, \mathcal{D}) := \min_{\mathbf{B} \in \mathcal{D}} \Delta_{\text{col}}(\mathbf{A}, \mathbf{B})$.

Lemma B.3 (Interleaved code proximity test [AHIV17]). *Let $\mathcal{C} \subseteq \mathbb{F}^L$ be a linear code of relative distance δ , and let $\mathcal{C}^{N_1} \subseteq \mathbb{F}^{N_1 \times L}$ be the corresponding interleaved code, i.e., every row is a codeword in $\mathcal{C}$. Consider the following test*

on an oracle matrix $\mathbf{C} \in \mathbb{F}^{N_1 \times L}$: the verifier samples $\mathbf{r} \leftarrow \mathbb{F}^{N_1}$, receives a word $\mathbf{u} \in \mathcal{C}$, samples a set $I \subseteq [L]$ of size t , and accepts only if $(\mathbf{u})_j = (\mathbf{r}^\top \mathbf{C})_j$ for every $j \in I$. If $\Delta_{\text{col}}(\mathbf{C}, \mathcal{C}^{N_1}) > e$, then any prover is accepted with probability at most

$$\left(1 - \frac{e}{L}\right)^t + \frac{e+1}{|\mathbb{F}|}.$$

Lemma B.4 (Knowledge soundness of the BaseFold batched evaluation protocol [NST25]). *Let $K = \text{poly}(\lambda)$ and $m = 2^\ell$. Suppose the BaseFold batch evaluation protocol accepts with probability at least $1/\text{poly}(\lambda)$ on public input commitments $\{\mathbf{c}_t\}_{t \in [K]}$, evaluation points $\{\mathbf{z}_t\}_{t \in [K]}$, and claimed values $\{y_t\}_{t \in [K]}$. Then there exists an extractor running in time $\text{poly}(m, K, \lambda)$ that outputs multilinear polynomials $f_t \in \mathbb{F}[X_1, \dots, X_\ell]$ consistent with the committed BaseFold oracles and satisfying $f_t(\mathbf{z}_t) = y_t$ for every $t \in [K]$, except with negligible probability.*

B.1 Completeness

Lemma B.5 (Completeness). *For every multilinear polynomial f , every evaluation point $\mathbf{z} \in \mathbb{F}^n$, and every honestly generated commitment to f , the honest prover convinces the verifier of the claimed value $y = f(\mathbf{z})$ with probability 1.*

Proof. Completeness follows directly from the descriptions of Quasar.Commit in Section 3.5, Quasar.Open above, and Quasar.Eval in Section 3.6.

B.2 Binding

The binding definition concerns the opening algorithm: a commitment should not be openable to two different polynomials. In the IOP model, this means that the same oracle matrix $\mathbf{C}$ cannot be accepted as an opening of two different multilinear polynomials. This follows directly from the distance of the QA code.

Lemma B.6 (Binding). *Assume that the QA code has relative distance δ_{QA} , and that Quasar.Open accepts an opening to f only if the row-wise QA encoding matrix $\mathbf{C}_f$ induced by f satisfies $\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}_f) \leq \frac{\delta_{\text{QA}}}{3}L$. Then Quasar satisfies binding in the IOP model.*

Proof. Suppose for contradiction that the same committed oracle $\mathbf{C}$ opens to two different multilinear polynomials f_0 and f_1 . Let $\mathbf{M}_0, \mathbf{M}_1 \in \mathbb{F}^{N_1 \times N_2}$ be their evaluation matrices over the Boolean hypercube, and let $\mathbf{C}_b(i, \cdot) := \text{QA.Enc}(\mathbf{M}_b(i, \cdot))$ for $b \in \{0, 1\}$ be the corresponding row-wise QA encodings. Since both openings are accepted, $\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}_0) \leq \frac{\delta_{\text{QA}}}{3}L$ and $\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}_1) \leq \frac{\delta_{\text{QA}}}{3}L$. By the triangle inequality, $\Delta_{\text{col}}(\mathbf{C}_0, \mathbf{C}_1) \leq \frac{2\delta_{\text{QA}}}{3}L$. On the other hand, since $f_0 \neq f_1$, there exists a row i^* such that $\mathbf{M}_0(i^*, \cdot) \neq \mathbf{M}_1(i^*, \cdot)$. Because the QA code is linear and has relative distance δ_{QA} , the two distinct QA codewords $\mathbf{C}_0(i^*, \cdot) = \text{QA.Enc}(\mathbf{M}_0(i^*, \cdot))$, $\mathbf{C}_1(i^*, \cdot) = \text{QA.Enc}(\mathbf{M}_1(i^*, \cdot))$ differ in at least $\delta_{\text{QA}}L$ positions. Therefore the matrices $\mathbf{C}_0$ and $\mathbf{C}_1$ differ in at least $\delta_{\text{QA}}L$ columns, i.e., $\Delta_{\text{col}}(\mathbf{C}_0, \mathbf{C}_1) \geq \delta_{\text{QA}}L$, which contradicts $\Delta_{\text{col}}(\mathbf{C}_0, \mathbf{C}_1) \leq \frac{2\delta_{\text{QA}}}{3}L$. Thus no commitment can be opened to two different polynomials in the IOP model.

B.3 Knowledge Soundness

We now prove knowledge soundness of the evaluation protocol. The proof has two steps. First, we show that any accepting evaluation proof is consistent with a unique row-wise QA-codeword matrix close to the committed oracle. Second, by rewinding the prover on sufficiently many independent row-combination challenges, the extractor recovers this matrix and hence the underlying multilinear polynomial.

Lemma B.7 (Soundness of the QA-encoding check). *Let ε_{BF} denote the knowledge-soundness error of the final batched BaseFold evaluation protocol. Suppose that Steps 10-12 of Quasar.Eval and the final batched BaseFold evaluation protocol accept. Then, except with probability at most $\varepsilon_{\text{BF}} + \frac{12n_2+2c-4}{|\mathbb{F}|}$, the vectors $\mathbf{u} = (\mathbf{u}_1 \parallel \dots \parallel \mathbf{u}_c)$ and $\mathbf{q} = (\mathbf{q}_1 \parallel \dots \parallel \mathbf{q}_c)$ are valid QA codewords. More precisely, $\mathbf{u} = \text{QA.Enc}(\mathbf{u}_1)$ and $\mathbf{q} = \text{QA.Enc}(\mathbf{q}_1)$.*

Proof. By the knowledge soundness of the final batched BaseFold evaluation protocol, except with probability ε_{BF} , all evaluation claims used at the points of the two sumchecks are consistent with the corresponding committed vectors. The evaluation claims for the public vectors $\mathbf{a}'_i$ are similarly consistent with their preprocessing commitments.

For every $i \in \{2, \dots, c\}$, define $\mathcal{E}_i(\mathbf{w}) := \text{WHT}(2^{-n_2} \mathbf{a}'_i \star \text{WHT}(\mathbf{w}))$. Consider first the proximity test and define

$$D_i^{(u)}(\mathbf{X}) := \tilde{u}_i(\mathbf{X}) - \widetilde{\mathcal{E}_i(\mathbf{u}_1)}(\mathbf{X}),$$

The verifier uses a challenge α to batch the $c - 1$ non systematic block relations. Define

$$Q_u(\mathbf{X}, A) := \sum_{i=2}^c A^{i-2} D_i^{(u)}(\mathbf{X}).$$

If any block relation for $\mathbf{u}$ is false, then at least one $D_i^{(u)}$ is nonzero. Since these polynomials are the coefficients of the distinct powers A^{i-2} , this implies $Q_u(\mathbf{X}, A) \not\equiv 0$. The polynomial Q_u is multilinear in $\mathbf{X}$ and has degree at most $c - 2$ in A . Hence, $\deg(Q_u) \leq n_2 + c - 2$. Because the evaluation point γ and the challenge α are sampled after the commitments have been fixed, the Schwartz–Zippel lemma gives

$$\Pr [Q_u(\gamma, \alpha) = 0] \leq \frac{n_2 + c - 2}{|\mathbb{F}|}.$$

Conditioned on $Q_u(\gamma, \alpha) \neq 0$, the initial claim of the $\mathbf{u}$ -branch is false. The sumcheck at the Step 11 contains the product of the WHT matrix, a public parameter polynomial, and an intermediate WHT polynomial. Its individual degree in every sumcheck variable is therefore at most 3. Since the outer sumcheck has n_2 rounds, its soundness error is at most $\frac{3n_2}{|\mathbb{F}|}$.

The terminal claim of the sumcheck at the Step 11 is reduced to evaluations of the systematic block $\mathbf{u}_1$ by the sumcheck at the Step 12. It contains the product

of the WHT matrix and a multilinear polynomial, and hence has individual degree at most 2. The soundness error of this sumcheck is therefore at most $\frac{2n_2}{|\mathbb{F}|}$.

It follows that the soundness error of the complete $\mathbf{u}$ -branch is at most $\frac{6n_2+c-2}{|\mathbb{F}|}$.

The $\mathbf{q}$ -branch is analyzed identically and contributes an error of at most $\frac{6n_2+c-2}{|\mathbb{F}|}$.

Applying a union bound, the overall failure probability is at most

$$\varepsilon_{\text{BF}} + \frac{12n_2 + 2c - 4}{|\mathbb{F}|}.$$

Lemma B.8 (Evaluation consistency). *Let ε_{BF} denote the knowledge-soundness error of the final batched BaseFold evaluation protocol. Suppose that a PPT adversary $\mathcal{A}$ causes the verifier to accept an evaluation proof on public input $(\mathbf{C}, \mathbf{z}, y)$ with probability at least $1/\text{poly}(\lambda)$. Then, except with negligible probability, there exists a multilinear polynomial $f' \in \mathbb{F}[X_1, \dots, X_n]$ such that $f'(\mathbf{z}) = y$. Moreover, if $\mathbf{M}' \in \mathbb{F}^{N_1 \times N_2}$ is the evaluation matrix of f' over $\{0, 1\}^n$, then, for every $\mathbf{i} \in \{0, 1\}^{n_1}$, $\Delta(\text{QA.Enc}(\mathbf{M}'(\mathbf{i}, \cdot)), \mathbf{C}(\mathbf{i}, \cdot)) \leq \frac{\delta_{\text{QA}} L}{3}$.*

Proof. Let $\varepsilon \geq 1/\text{poly}(\lambda)$ be the acceptance probability of $\mathcal{A}$. Let $\mathbf{r} \leftarrow \mathbb{F}^{N_1}$ be the verifier's random row-combination vector, let $I = \{i_0, \dots, i_{t-1}\} \subseteq [L]$ be the sampled set of $t = \Theta(\lambda)$ columns, and let $\ell = \lceil \log L \rceil$.

We first extract the vectors committed through BaseFold. By Lemma B.1, for an inverse-polynomial fraction of the transcript prefixes, the conditional acceptance probability of the remaining protocol is still inverse-polynomial. For each such prefix, the knowledge soundness of the final batched BaseFold evaluation protocol implies that, except with probability ε_{BF} , the extractor obtains well-defined vectors $\mathbf{u}_1, \dots, \mathbf{u}_c, \mathbf{q}_1, \dots, \mathbf{q}_c$, and all terminal evaluation claims made about these vectors are consistent with their committed oracles. The evaluations of the public QA-parameter vectors are likewise consistent with their preprocessing commitments.

By Lemma B.7, the extracted vectors $\mathbf{u} = (\mathbf{u}_1 \parallel \dots \parallel \mathbf{u}_c)$ and $\mathbf{q} = (\mathbf{q}_1 \parallel \dots \parallel \mathbf{q}_c)$ are valid QA codewords, except with $\varepsilon_{\text{BF}} + \frac{12n_2+2c-4}{|\mathbb{F}|}$. Thus, we have $\mathbf{u} = \text{QA.Enc}(\mathbf{v})$ and $\mathbf{q} = \text{QA.Enc}(\mathbf{p})$.

We next analyze the selector-sumchecks. Consider first the proximity test. For each $j = 0, \dots, t-1$, define $e_j^{(u)} := (\mathbf{u})_{i_j} - \langle \mathbf{r}, \mathbf{C}(\cdot, i_j) \rangle$. The selector-sumcheck verifies $\sum_{j=0}^{t-1} \tau^j e_j^{(u)} = 0$, where $\tau \leftarrow \mathbb{F}$ is sampled randomly. If some $e_j^{(u)}$ is nonzero, the left-hand side is the evaluation at τ of a nonzero univariate polynomial of degree at most $t-1$. Therefore, by the Schwartz–Zippel lemma, this identity holds with probability at most $\frac{t-1}{|\mathbb{F}|}$. The same argument applies to the evaluation check. Since the two branches use the same challenge τ , if either branch contains an incorrect sampled-position relation, both batching identities accept with probability at most $(t-1)/|\mathbb{F}|$.

Each selector-sumcheck has ℓ rounds and an individual degree 2. Its sumcheck soundness error is therefore at most $2\ell/|\mathbb{F}|$. For the two selector-sumchecks, the

total error is at most $\frac{4\ell}{|\mathbb{F}|}$. Consequently, with probability at most

$$\varepsilon_{\text{sel}} := \frac{t - 1 + 4\ell}{|\mathbb{F}|},$$

for every $j = 0, \dots, t - 1$, $(\mathbf{u})_{i_j} = \langle \mathbf{r}, \mathbf{C}(\cdot, i_j) \rangle$ and $(\mathbf{q})_{i_j} = \langle \mathbf{z}_L, \mathbf{C}(\cdot, i_j) \rangle$ hold.

We now prove that $\mathbf{C}$ is close to the interleaved QA code. Let $\mathcal{C}_{\text{QA}} \subseteq \mathbb{F}^L$ denote the QA code and let $\mathcal{C}_{\text{QA}}^{N_1}$ denote its N_1 -fold interleaving. Suppose that $\Delta_{\text{col}}(\mathbf{C}, \mathcal{C}_{\text{QA}}^{N_1}) > \frac{\delta_{\text{QA}}L}{3}$. Since $\mathbf{u}$ is a QA codeword, Lemma B.3 implies that the probability that the sampled-column consistency test accepts is at most

$$\left(1 - \frac{\delta_{\text{QA}}}{3}\right)^t + \frac{e + 1}{|\mathbb{F}|}.$$

This is negligible under our parameter choices. This contradicts the assumed inverse-polynomial acceptance probability. Hence there exists $\mathbf{C}' \in \mathcal{C}_{\text{QA}}^{N_1}$ such that $\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}') \leq \frac{\delta_{\text{QA}}L}{3}$.

This matrix $\mathbf{C}'$ is unique. Indeed, if $\mathbf{D}, \mathbf{D}' \in \mathcal{C}_{\text{QA}}^{N_1}$ are distinct, then at least one row contains two distinct QA codewords. Therefore $\Delta_{\text{col}}(\mathbf{D}, \mathbf{D}') \geq \delta_{\text{QA}}L$. On the other hand, if both matrices were within column distance $\frac{\delta_{\text{QA}}L}{3}$ of $\mathbf{C}$, then $\Delta_{\text{col}}(\mathbf{D}, \mathbf{D}') \leq \frac{2\delta_{\text{QA}}L}{3}$, which is a contradiction.

Since every row of $\mathbf{C}'$ is a QA codeword and QA.Enc is systematic, for every $\mathbf{i} \in \{0, 1\}^{n_1}$ there exists a unique message row $\mathbf{M}'(\mathbf{i}, \cdot) \in \mathbb{F}^{N_2}$ such that $\mathbf{C}'(\mathbf{i}, \cdot) = \text{QA.Enc}(\mathbf{M}'(\mathbf{i}, \cdot))$. Let $\mathbf{M}' \in \mathbb{F}^{N_1 \times N_2}$ be the matrix formed by these message rows.

Moreover, every disagreement between a row of $\mathbf{C}$ and the corresponding row of $\mathbf{C}'$ occurs in a column on which the two matrices differ. Hence, for every $\mathbf{i} \in \{0, 1\}^{n_1}$,

$$\Delta(\mathbf{C}(\mathbf{i}, \cdot), \text{QA.Enc}(\mathbf{M}'(\mathbf{i}, \cdot))) \leq \Delta_{\text{col}}(\mathbf{C}, \mathbf{C}') \leq \frac{\delta_{\text{QA}}L}{3}.$$

We next prove that the extracted codeword $\mathbf{u}$ encodes the row combination of $\mathbf{M}'$, i.e. $\mathbf{u} = \mathbf{r}^\top \mathbf{C}'$. Suppose otherwise. Both vectors are QA codewords. Therefore $\Delta(\mathbf{u}, \mathbf{r}^\top \mathbf{C}') \geq \delta_{\text{QA}}L$. Furthermore, $\Delta(\mathbf{r}^\top \mathbf{C}, \mathbf{r}^\top \mathbf{C}') \leq \Delta_{\text{col}}(\mathbf{C}, \mathbf{C}') \leq \frac{\delta_{\text{QA}}L}{3}$, because a linear combination can differ only at a column at which $\mathbf{C}$ and $\mathbf{C}'$ differ. The triangle inequality therefore gives $\Delta(\mathbf{u}, \mathbf{r}^\top \mathbf{C}) \geq \frac{2\delta_{\text{QA}}L}{3}$. A uniformly sampled set I of size t avoids all these disagreements with probability at most

$$\left(1 - \frac{2\delta_{\text{QA}}}{3}\right)^t.$$

Outside the selector-sumcheck failure event, this contradicts $(\mathbf{u})_j = (\mathbf{r}^\top \mathbf{C})_j$ for every $j \in I$. Thus $\mathbf{u} = \mathbf{r}^\top \mathbf{C}' = \text{QA.Enc}(\mathbf{r}^\top \mathbf{M}')$. Since the code is systematic, its first block satisfies $\mathbf{u}_1 = \mathbf{r}^\top \mathbf{M}'$.

An identical argument applies to $\mathbf{q}$. We claim that $\mathbf{q} = \mathbf{z}_L^\top \mathbf{C}'$. If this equality were false, then $\mathbf{q}$ and $\mathbf{z}_L^\top \mathbf{C}' = \text{QA.Enc}(\mathbf{z}_L^\top \mathbf{M}')$ would be distinct QA codewords.

Consequently, $\Delta(\mathbf{q}, \mathbf{z}_L^\top \mathbf{C}) \geq \frac{2\delta_{\text{QA}}L}{3}$. The probability that the sampled set I avoids all disagreements would again be at most $\left(1 - \frac{2\delta_{\text{QA}}}{3}\right)^t$, contradicting the accepted selector-sumcheck. Therefore $\mathbf{q} = \mathbf{z}_L^\top \mathbf{C}' = \text{QA.Enc}(\mathbf{z}_L^\top \mathbf{M}')$. By systematicity, $\mathbf{q}_1 = \mathbf{z}_L^\top \mathbf{M}'$.

The final batched BaseFold evaluation protocol contains the claim $\tilde{q}_1(\mathbf{z}_R) = y$. Its knowledge soundness guarantees that this value is the evaluation of the extracted vector $\mathbf{q}_1$. Hence $y = \tilde{q}_1(\mathbf{z}_R) = \widetilde{\mathbf{z}_L^\top \mathbf{M}'(\mathbf{z}_R)}$.

Let f' be the unique multilinear polynomial whose evaluation table on $\{0, 1\}^n$ is $\mathbf{M}'$. By the standard decomposition of multilinear evaluations, $\widetilde{\mathbf{z}_L^\top \mathbf{M}'(\mathbf{z}_R)} = f'(\mathbf{z})$. Therefore $f'(\mathbf{z}) = y$.

For completeness, the total failure probability used above is bounded by

$$\begin{aligned} \varepsilon_{\text{cons}} \leq \varepsilon_{\text{BF}} + \frac{12n_2 + 2c - 4}{|\mathbb{F}|} + \frac{t - 1 + 4\ell}{|\mathbb{F}|} \\ + \left(1 - \frac{\delta_{\text{QA}}}{3}\right)^t + \frac{\delta_{\text{QA}}L/3 + 1}{|\mathbb{F}|} + 2 \left(1 - \frac{2\delta_{\text{QA}}}{3}\right)^t. \end{aligned}$$

For $t = \Theta(\lambda)$, constant relative distance δ_{QA} , and the stated field-size assumptions, this quantity is negligible. The claimed evaluation consistency and row-wise proximity therefore follow.

Lemma B.9 (Knowledge soundness). *Suppose a PPT adversary $\mathcal{A}$ causes the verifier to accept an evaluation proof on public input $(\mathbf{C}, \mathbf{z}, y)$ with probability at least $1/\text{poly}(\lambda)$. Then there exists an extractor Ext running in time $\text{poly}(N, \lambda)$ that outputs a multilinear polynomial $f \in \mathbb{F}[X_1, \dots, X_n]$ such that $f(\mathbf{z}) = y$ and $\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}_f) \leq \frac{\delta_{\text{QA}}}{3} cN_2$, where $\mathbf{C}_f$ is the row-wise QA encoding matrix induced by f . Equivalently, there exists an opening hint $\mathbf{h} = (\mathbf{M}_f, \mathbf{C}, \mathcal{T})$ such that $\text{Quasar.Open}(\text{pp}, \mathbf{c}, f, \mathbf{h}) = 1$.*

Proof. For a row-combination challenge $\mathbf{r} \in \mathbb{F}^{N_1}$, let $A(\mathbf{r}) = 1$ denote the event that the verifier accepts and that the extraction guaranteed by Lemma B.8 succeeds. By the assumed acceptance probability and Lemma B.1,

$$\Pr_{\mathbf{r} \leftarrow \mathbb{F}^{N_1}} [A(\mathbf{r}) = 1] \geq \frac{1}{\text{poly}(\lambda)}.$$

Let $\mathcal{M} = \mathbb{F}^{N_1}$, and define $\Phi(\mathbf{r}_1, \dots, \mathbf{r}_i) = 1$ if and only if $\mathbf{r}_1, \dots, \mathbf{r}_i$ are linearly independent. For every $i < N_1$, conditioned on $\Phi(\mathbf{r}_1, \dots, \mathbf{r}_i) = 1$, a uniformly random vector $\mathbf{r}_{i+1} \leftarrow \mathbb{F}^{N_1}$ is linearly independent of the previous vectors with probability $1 - \frac{|\mathbb{F}|^i}{|\mathbb{F}|^{N_1}}$. Thus Φ satisfies the condition of Lemma B.2. Applying that lemma with $N = N_1$, the extractor obtains linearly independent challenges $\mathbf{r}_1, \dots, \mathbf{r}_{N_1} \in \mathbb{F}^{N_1}$ such that $A(\mathbf{r}_j) = 1$ for every $j \in [N_1]$ except with negligible probability. For each challenge $\mathbf{r}_j$, the final batched BaseFold extractor obtains the vectors $\mathbf{u}_1^{(j)}, \dots, \mathbf{u}_c^{(j)}$. Define

$$\mathbf{w}_j := \mathbf{u}_1^{(j)} \parallel \dots \parallel \mathbf{u}_c^{(j)} \in \mathbb{F}^L.$$

The QA-encoding check guarantees that $\mathbf{w}_j$ is a valid QA codeword.

By Lemma B.8, there is a unique row-wise QA-codeword matrix $\mathbf{C}' \in \mathbb{F}^{N_1 \times L}$ such that $\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}') \leq \frac{\delta_{\text{QA}}}{3}L$, and every successful extraction satisfies $\mathbf{w}_j = \mathbf{r}_j^\top \mathbf{C}'$. The same matrix $\mathbf{C}'$ applies to all extracted transcripts because it is the unique codeword matrix within this radius of the fixed oracle $\mathbf{C}$.

Let $\mathbf{R} \in \mathbb{F}^{N_1 \times N_1}$ be the matrix whose j -th row is $\mathbf{r}_j^\top$, and let $\mathbf{W} \in \mathbb{F}^{N_1 \times L}$ be the matrix whose j -th row is $\mathbf{w}_j$. Then

$$\mathbf{W} = \mathbf{R}\mathbf{C}'.$$

Since the vectors $\mathbf{r}_1, \dots, \mathbf{r}_{N_1}$ are linearly independent, $\mathbf{R}$ is invertible. Hence the extractor recovers

$$\mathbf{C}' = \mathbf{R}^{-1}\mathbf{W}.$$

Since every row of $\mathbf{C}'$ is a QA codeword and QA.Enc is systematic, the extractor recovers the unique matrix $\mathbf{M}' \in \mathbb{F}^{N_1 \times N_2}$ satisfying $\mathbf{C}'(\mathbf{i}, \cdot) = \text{QA.Enc}(\mathbf{M}'(\mathbf{i}, \cdot))$ for every $\mathbf{i} \in \{0, 1\}^{n_1}$.

Let f be the unique multilinear polynomial whose evaluations over $\{0, 1\}^n$ are given by $\mathbf{M}'$. For any one of the successful transcripts, Lemma B.8 gives $f(\mathbf{z}) = y$. Moreover, $\mathbf{C}_f = \mathbf{C}'$ and therefore

$$\Delta_{\text{col}}(\mathbf{C}, \mathbf{C}_f) \leq \frac{\delta_{\text{QA}}}{3}L = \frac{\delta_{\text{QA}}}{3}cN_2.$$

Taking $\mathbf{h} = (\mathbf{M}', \mathbf{C}, \mathcal{T})$, where $\mathcal{T}$ is the Merkle tree corresponding to $\mathbf{C}$, gives Quasar.Open(pp, c, f, h) = 1. Thus f is a valid witness for the evaluation relation.

The extractor invokes the verifier and the underlying BaseFold extractor polynomially many times, applies Lemma B.2, and performs linear algebra on an $N_1 \times N_1$ matrix. Its total running time is therefore $\text{poly}(N, \lambda)$.

Proof (Proof of Theorem 3.1). Completeness follows from Lemma B.5, and binding in the IOP model follows from Lemma B.6. Knowledge soundness in the IOP model follows from Lemma B.9.

It remains to lift the preceding IOP guarantees to the implemented protocol. For every IOP oracle message, the prover first sends a vector commitment to that oracle; the verifier then sends the next public-coin challenge; and, after the IOP query positions have been sampled, the prover opens precisely those positions with authentication proofs. Thus, Quasar.Eval is an instance of the interactive BCS, or IBCS, compiler applied to the public-coin IOP analyzed above [CDGS23].

Condition on the event $\text{GoodDist} := \{\delta(\mathcal{C}_{\text{QA}}) \geq \delta_{\text{QA}}\}$. By Lemmas B.6 and B.9, the underlying IOP is binding and knowledge sound with error at most ε_{IOP} under this condition. The security theorem for the IBCS compiler then gives

$$\varepsilon_{\text{IBCS}} \leq \varepsilon_{\text{IOP}} + \varepsilon_{\text{VC}},$$

where ε_{VC} is negligible under the vector-commitment security assumption. In particular, the IBCS extractor interfaces directly with the IOP extractor of

Table 4: Concrete Quasar parameters and security bounds. All rows use Mersenne127, $N_1 = 2^8$ matrix rows, QA inverse rate $c = 2$, 249 BaseFold repetitions, and a 128-bit classical binding target for the BLAKE2s Merkle vector commitment. We write $\lambda_x = -\log_2 \varepsilon_x$. The last column applies the complete union bound in Equation (2).

k	N_2	L	δ_{QA}	t	λ_{dist}	λ_{samp}	λ_{fid}	λ_{BF}	λ_{total}
20	2^{12}	2^{13}	0.444951	445	103.0	103.1	116.1	103.3	101.5
21	2^{13}	2^{14}	0.441024	450	103.0	103.2	115.4	103.3	101.6
22	2^{14}	2^{15}	0.437098	454	103.0	103.1	114.6	103.3	101.6
23	2^{15}	2^{16}	0.433173	458	103.0	103.0	113.7	103.3	101.5
24	2^{16}	2^{17}	0.429248	463	103.0	103.1	112.8	103.3	101.6
25	2^{17}	2^{18}	0.425324	467	103.0	103.0	111.8	103.3	101.5
26	2^{18}	2^{19}	0.421401	472	103.0	103.1	110.8	103.3	101.5
27	2^{19}	2^{20}	0.417478	477	103.0	103.1	109.8	103.3	101.6
28	2^{20}	2^{21}	0.413556	482	103.0	103.1	108.9	103.3	101.6
29	2^{21}	2^{22}	0.409635	487	103.0	103.2	107.9	103.3	101.6

Lemma B.9; it does not rewind the authenticated-query phase until every column has been collected. Finally, $\Pr[\neg \text{GoodDist}] \leq \varepsilon_{\text{dist}}$, so removing the conditioning yields

$$\varepsilon_{\text{Quasar}} \leq \varepsilon_{\text{dist}} + \varepsilon_{\text{IOP}} + \varepsilon_{\text{VC}}.$$

This proves Theorem 3.1. The prover time, verifier time, and proof-size bounds follow from Appendix A.

C Parameters

For the concrete experiments, we set the principal statistical components to approximately 103 bits, leaving sufficient slack for their union. For each row length N_2 , the setup algorithm samples $\mathcal{C}_{\text{QA}} \leftarrow \text{QA.Gen}$ and selects δ_{QA} so that

$$\varepsilon_{\text{dist}} := \Pr[\delta(\mathcal{C}_{\text{QA}}) < \delta_{\text{QA}}] \leq 2^{-103}.$$

It then sets

$$t = \left\lceil \frac{103}{-\log_2(1 - \delta_{\text{QA}}/3)} \right\rceil,$$

which guarantees $\varepsilon_{\text{samp}} := (1 - \delta_{\text{QA}}/3)^t \leq 2^{-103}$.

The final batched BaseFold opening uses the rate-1/2 unique-decoding setting. With $s_{\text{BF}} = 249$ query repetitions, the BaseFold query error is

$$\varepsilon_{\text{BF}} \leq \left(\frac{3}{4}\right)^{249}, \lambda_{\text{BF}} := -\log_2 \varepsilon_{\text{BF}} \geq 249 \log_2(4/3) = 103.34.$$

This replaces the previous choice of 241 repetitions, which provides only 100.02 bits for this component.

Let $\ell = \lceil \log_2 L \rceil$. Combining the explicit field-size terms in Appendix B, define

$$\varepsilon_{\text{fld}} := \frac{12n_2 + 2c - 4 + (t - 1 + 4\ell) + (\delta_{\text{QA}}L/3 + 1)}{|\mathbb{F}|}.$$

The remaining sampled-column term is $\varepsilon_{\text{aux}} := 2(1 - 2\delta_{\text{QA}}/3)^t$. Therefore the complete error accounting is

$$\begin{aligned} \varepsilon_{\text{Quasar}} &\leq \varepsilon_{\text{dist}} + \varepsilon_{\text{samp}} + \varepsilon_{\text{aux}} + \varepsilon_{\text{fld}} \\ &\quad + \varepsilon_{\text{BF}} + \varepsilon_{\text{VC}}. \end{aligned} \tag{2}$$

Using $\varepsilon_{\text{VC}} \leq 2^{-128}$ as the classical binding target for the BLAKE2s-based Merkle vector commitment, the right-hand side is at most $2^{-101.5}$ for every parameter row in Table 4. Hence all reported parameter sets provide at least 100 bits of concrete security under the stated assumptions.

D Comparison with Concurrent QAFold

Concurrent to our work, SwitchFold [CYY26] proposes a code-agnostic PCS framework based on recursive code switching and instantiates it with QA codes to obtain QAFold. Both QAFold and Quasar use QA codes over large prime fields and achieve $O(\lambda \log^2 N)$ proof size and verifier time. Their main difference is therefore not the asymptotic succinctness, but how the QA-code consistency relation is reduced to succinctly verifiable claims.

Protocol comparison. SwitchFold recursively switches between a sequence of codes with geometrically decreasing message lengths. At every level j , the switching procedure not only produces a witness for the next code, but also leaves an MLE claim on the generator matrix. Thus, a PCS opening generates generator-matrix claims at multiple recursive levels. Rather than resolving these claims immediately, SwitchFold uses an accumulation scheme to combine them across openings and postpones their verification to a separate **Decide** phase. The **Decide** procedure then recursively opens the accumulated generator-matrix claims.

QAFold exploits the structure of QA generator matrices to simplify this generic procedure. Nevertheless, the main recursive machinery remains: QAFold still performs multi-level code switching, generates and accumulates generator-matrix MLE claims, and eventually invokes the recursive **Decide** procedure.

In contrast, Quasar verifies the native QA encoding relation directly. Its QA-specific reduction eliminates the intermediate encoding witnesses and reduces the encoding check to two structured sumchecks, followed by the batched BaseFold opening already used by the PCS. Consequently, it does not generate generator-matrix MLE obligations and requires neither cross-instance accumulation nor a separate recursive **Decide** phase. Hence, when the underlying code is fixed to QA, Quasar trades the code agnosticity of SwitchFold for a substantially simpler prover path while retaining the same $O(\lambda \log^2 N)$ asymptotic proof size and verifier time.

Experimental comparison. We do not report a direct wall-clock comparison because the QAFold experiments use different hardware and CPU parallelism from ours. Moreover, the implementation URL provided by the paper was not publicly accessible at the time of writing, preventing us from reproducing QAFold under our experimental setting.