Twist and Shout via logup*

Georg Wiese
Powdr Labs
georg@powdrlabs.com

October 30, 2025

Abstract

We explore how to bring the Twist and Shout memory checking arguments to hash-based proof systems by leveraging logup*. Twist and Shout significantly reduce commitment costs for memory checking, especially for small memories. However, they rely on efficient commitments to sparse polynomials, which is challenging for hash-based schemes. We show that using ideas from logup*, these protocols can be adapted to work efficiently in the hash-based setting, enabling cheaper lookups for large tables and low-cost read/write memory arguments. The core insight is that there is an implicit commitment scheme for matrices with one-hot rows in logup*, which can be plugged into Twist and Shout with minor modifications.

1 Introduction

Twist and Shout [5] are recent memory checking arguments. Shout is an indexed lookup argument, or — equivalently — a read-only memory argument. Twist is a read/write memory argument which significantly reduces commitment costs compared to arguments based on offline memory checking [1], as used in zkVMs such as OpenVM[1] and SP1[2]. This is especially true for *small* memories, for example RISC-V registers. Both protocols rely on being able to commit to *sparse* polynomials efficiently, which is the case for commitment schemes based on elliptic curves, but not for hash-based commitment schemes.

Logup* [6] is also a recent read-only memory argument with similar properties to Shout. In particular, the prover only needs to commit to values *once*, not once for every memory access. Since it requires the prover to commit to extra data proportional to the table size, it is most efficient for small tables.

In this work, we show how to combine Twist and Shout with techniques from logup*. The resulting read-only memory argument has lower commitment costs than logup* alone when the memory is large. The resulting read/write memory argument has significantly lower commitment costs than prior techniques based on offline memory checking, especially for small memories.

2 Notation and Preliminaries

Given a field $\mathbb{F}$, we denote by $\mathbb{F}_{\text{ext}}$ its cryptographically large extension field. Given a matrix $M \in \mathbb{F}^{n \times m}$, we denote by $\widetilde{M} : \mathbb{F}^{\log(n)} \times \mathbb{F}^{\log(m)} \to \mathbb{F}$ its multilinear extension, i.e.:

[1] `https://github.com/openvm-org/openvm`
[2] `https://github.com/succinctlabs/sp1`

$$\widetilde{M}(x_{\text{row}}, x_{\text{col}}) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} M[i,j] \cdot \widetilde{\text{eq}}(\texttt{bits}(i), x_{\text{row}}) \cdot \widetilde{\text{eq}}(\texttt{bits}(j), x_{\text{col}}) \tag{1}$$

where $\texttt{bits} : \mathbb{N}_0 \to \mathbb{F}^{\log(n)}$ is the function that maps a natural number to its binary representation and $\widetilde{\text{eq}}$ is the multilinear extension of the equality function, defined as:

$$\widetilde{\text{eq}}(a_0, \ldots, a_{k-1}, b_0, \ldots, b_{k-1}) = \prod_{i=0}^{k-1} (a_i \cdot b_i + (1 - a_i) \cdot (1 - b_i)) \tag{2}$$

As a special case, for a vector $v \in \mathbb{F}^n$, we denote its multilinear extension by $\widetilde{v} : \mathbb{F}^{\log(n)} \to \mathbb{F}$. We will often use the notation of vectors and matrices and mean their multilinear extensions. In particular, when we talk about committing to a matrix or vector, we mean committing to the polynomial given by its multilinear extension.

Many interactive protocols involving multilinear extensions are built on top of the sum-check protocol [3]. For a multivariate polynomial $f(x_1, \ldots, x_k)$, the sum-check protocol allows the prover to convince the verifier of the value of:

$$s := \sum_{(b_1, \ldots, b_k) \in \{0,1\}^k} f(b_1, \ldots, b_k) \tag{3}$$

At the end of the protocol, the verifier is left with a claim for $f(r_1, \ldots, r_k)$ for random $r_i \in \mathbb{F}_{\text{ext}}$. This claim can be checked by the verifier either by evaluating f directly (e.g., if f is known in advance and there exists an algorithm to evaluate it in logarithmic time), via a polynomial commitment scheme, or by invoking another protocol.

3 Logup* and the pushforward

Logup* [6] is an indexed lookup argument. Its main improvement over [4] is that the prover does not have to commit to polynomials encoding the looked-up values. Instead, whenever the verifier wishes to evaluate these polynomials, the prover and verifier engage in a protocol that forces the prover (w.h.p.) to send the correct evaluation. We refer to such polynomials as *virtual* polynomials.

A crucial component of logup* is the notion of a pushforward. Given a vector $A \in \mathbb{F}^n$ and a mapping $I : \{0, \ldots, n-1\} \to \{0, \ldots, m-1\}$, the pushforward $I_*A \in \mathbb{F}^m$ is a vector defined as follows:

$$I_*A[j] = \sum_{i | I[i] = j} A[i] \tag{4}$$

Well-formedness of a pushforward can be proven using the GKR protocol [2], using an argument similar to [4]. This subprotocol is not presented in detail here. We refer to Section 4 of [6].

Logup* asks the prover to commit to $I_*\text{eq}_r$ where I indicates the indices and $\text{eq}_r \in \mathbb{F}_{\text{ext}}^n$ is the vector of the n Lagrange polynomials evaluated at point $r \in \mathbb{F}_{\text{ext}}^{\log(n)}$, i.e., $\text{eq}_r[i] = \widetilde{\text{eq}}(\texttt{bits}(i), r)$. That is, $I_*\text{eq}_r$ is a vector of m extension field elements. In cases where $n \gg m$, the commitment cost is significantly reduced compared to [4]. For the full protocol, we refer to [6].

4 Committing to matrices with one-hot rows

Consider a matrix $M \in \mathbb{F}^{n \times m}$ such that exactly one entry in each row is 1 and all other entries are 0. Let $M_{\text{dense}} \in \mathbb{F}^n$ be a vector of field elements indicating the nonzero index for each row. We present a protocol to evaluate $\widetilde{M}$ given oracle access to $\widetilde{M}_{\text{dense}}$.

Let $P := M_{\text{dense}*}\text{eq}_{r_{\text{row}}} \in \mathbb{F}_{\text{ext}}^m$. We claim that:

$$\widetilde{M}(r_{\text{row}}, r_{\text{col}}) = \widetilde{P}(r_{\text{col}}) \tag{5}$$

This follows from the definition of the pushforward, $\text{eq}_{r_{\text{row}}}$ and M_{dense}:

$$\begin{aligned}
\widetilde{P}(r_{\text{col}}) &= \langle P, \text{eq}_{r_{\text{col}}} \rangle \\
&= \sum_{j=0}^{m-1} P[j] \cdot \widetilde{\text{eq}}(\texttt{bits}(j), r_{\text{col}}) \\
&= \sum_{j=0}^{m-1} \Big(\sum_{i | M_{\text{dense}}[i]=j} \widetilde{\text{eq}}(\texttt{bits}(i), r_{\text{row}}) \Big) \cdot \widetilde{\text{eq}}(\texttt{bits}(j), r_{\text{col}}) \\
&= \sum_{(i,j) | M[i,j]=1} \widetilde{\text{eq}}(\texttt{bits}(i), r_{\text{row}}) \cdot \widetilde{\text{eq}}(\texttt{bits}(j), r_{\text{col}}) \\
&= \sum_{(i,j) | M[i,j]=1} \widetilde{\text{eq}}(\texttt{bits}(i) \mid\mid \texttt{bits}(j), r_{\text{row}} \mid\mid r_{\text{col}}) \\
&= \widetilde{M}(r_{\text{row}}, r_{\text{col}})
\end{aligned}$$

This provides a commitment scheme for matrices with one-hot rows:

1. Instead of committing to $M \in \mathbb{F}^{n \times m}$, the prover commits to the multilinear extension of $M_{\text{dense}} \in \mathbb{F}^n$ using a standard multilinear polynomial commitment scheme.
2. To evaluate $\widetilde{M}$ at $(r_{\text{row}}, r_{\text{col}})$, the prover and verifier run the protocol from Figure 1.

4.1 Batching pushforward commitments

The commitment costs for opening multiple multilinear extensions of matrices with one-hot rows can be reduced by batching. For $1 \le i \le d$, let $M_{\text{dense}}^{(i)} \in \mathbb{F}^n$ be the dense encoding of the i-th matrix $M^{(i)} \in \mathbb{F}^{n \times m}$ with one-hot rows. For simplicity, we assume that d is a power of two. Let $M_{\text{dense}}^{(*)} := M_{\text{dense}}^{(1)} \mid\mid \cdots \mid\mid M_{\text{dense}}^{(d)} \in \mathbb{F}^{n \cdot d}$ be the concatenation. Note that its multilinear extension $\widetilde{M}_{\text{dense}}^{(*)}$ can be evaluated efficiently given oracle access to $\widetilde{M}_{\text{dense}}^{(i)}$ for $1 \le i \le d$.

$M_{\text{dense}}^{(*)}$ describes a matrix $M^{(*)} \in \mathbb{F}^{n \cdot d \times m}$ which is the concatenation of the original matrices along the row dimension. The key observation is that $\widetilde{M^{(i)}}(x_{\text{row}}, x_{\text{col}}) = \widetilde{M^{(*)}}(x_{\text{row}}, \texttt{bits}(i), x_{\text{col}})$. Therefore, d claims for $\widetilde{M^{(i)}}$ are equivalent to d claims for $\widetilde{M^{(*)}}$.

To validate evaluation claims $\widetilde{M^{(i)}}(r_{\text{row}}^{(i)}, r_{\text{col}}^{(i)}) = \widetilde{M^{(*)}}(r_{\text{row}}^{(i)}, \texttt{bits}(i), r_{\text{col}}^{(i)})$ (for $1 \le i \le d$), the prover and verifier run the following protocol:

1. Using a standard technique (e.g., [7], section 4.5.2), the d claims about $\widetilde{M^{(*)}}$ are reduced to a single claim at an evaluation point $r \in \mathbb{F}_{\text{ext}}^{\log(n)+\log(d)+\log(m)}$.

Commitment Scheme for matrices with one-hot rows (opening protocol)

Input: Commitment to $M_{\text{dense}} \in \mathbb{F}^n$ encoding a matrix with one-hot rows $M \in \mathbb{F}^{n \times m}$ and evaluation points $r_{\text{row}} \in \mathbb{F}_{\text{ext}}^{\log(n)}$, $r_{\text{col}} \in \mathbb{F}_{\text{ext}}^{\log(m)}$.

Output: $\widetilde{M}(r_{\text{row}}, r_{\text{col}}) \in \mathbb{F}_{\text{ext}}$.

Protocol:

1. Let $P = M_{\text{dense}*}\text{eq}_{r_{\text{row}}} \in \mathbb{F}_{\text{ext}}^m$ be the pushforward $\text{eq}_{r_{\text{row}}}$ along M_{dense}.
2. Prover sends a commitment to $\widetilde{P}$ and a claimed value for $\widetilde{P}(r_{\text{col}})$.
3. Prover and verifier run the GKR protocol to prove well-formedness of P (Section 4 of [6]). This results in prover claims for $\widetilde{P}(r_1)$, $\widetilde{\text{eq}}(r_2, r_{\text{row}})$, and $\widetilde{M}_{\text{dense}}(r_3)$.
4. The verifier evaluates $\widetilde{\text{eq}}(r_2, r_{\text{row}})$ in logarithmic time; claims for $\widetilde{P}$ and $\widetilde{M}_{\text{dense}}$ are verified via the polynomial commitments.

Figure 1: Protocol for evaluating the multilinear extension of a matrix $M \in \mathbb{F}^{n \times m}$ with one-hot rows at a random point $(r_{\text{row}}, r_{\text{col}}) \in \mathbb{F}_{\text{ext}}^{\log(n)} \times \mathbb{F}_{\text{ext}}^{\log(m)}$, using a commitment to the dense encoding $M_{\text{dense}} \in \mathbb{F}^n$. This commitment scheme is implicit in logup* [6] and builds upon its subprotocol to prove well-formedness of a pushforward.

2. The single claim is verified using the algorithm from Figure 1. As a slight modification, there is no commitment to $\widetilde{M}_{\text{dense}}^{(*)}$. Instead, the verifier reduces its evaluation to queries to $\widetilde{M}_{\text{dense}}^{(i)}$ for $1 \leq i \leq d$ and asks the prover to open the corresponding commitments.

As part of step 2, the prover commits to a *single* pushforward $P \in \mathbb{F}_{\text{ext}}^m$, reducing the commitment cost by a factor of d over the naive approach of running the opening protocol d times.

5 Twist and Shout via logup*

In this section, we show how to adapt Shout and Twist using the commitment scheme extracted from logup* (Section 4). Both Twist and Shout are built around matrices with one-hot rows. In the original protocols, these matrices are committed to using commitment schemes based on elliptic curves, which allow for efficient commitments to sparse polynomials. Furthermore, [5] provides a subprotocol to prove the one-hot property of these matrices.

We propose instead to use the commitment scheme from Section 4.

5.1 Shout via logup*

Shout is a read-only memory checking argument. Let K be the memory size and T the number of memory accesses. In essence, the protocol combines two insights:

- If memory addresses are encoded as matrices with one-hot rows, then the vector of read values can be expressed as a matrix-vector product between the T-by-K address matrix and the K-dimensional memory value vector. The vector of read values can be virtual and evaluated using sum-checks.

- For large memories, the address one-hot vectors can be decomposed into d smaller one-hot vectors of dimension $\sqrt[d]{K}$, allowing for more efficient commitment openings and smaller sum-checks.

The protocol is best summarized in Figure 7 of [5]. The only commitments required are to the d address matrices with one-hot rows $\mathrm{ra}^{(i)} \in \mathbb{F}^{T \times \sqrt[d]{K}}$ and to the memory values $\mathrm{Val} \in \mathbb{F}^K$. We adapt Shout to hash-based commitment schemes by committing to the *dense* address vectors $\mathrm{ra}^{(i)}_{\mathrm{dense}} \in \mathbb{F}^T$ and using the batched opening protocol from Section 4.1 to evaluate $\widetilde{\mathrm{ra}}^{(i)}$.

We obtain a read-only memory argument with the following commitment costs:

- d read address vectors $\mathrm{ra}^{(i)}_{\mathrm{dense}} \in \mathbb{F}^T$.
- the memory content vector $\mathrm{Val} \in \mathbb{F}^K$.
- one batched pushforward $P \in \mathbb{F}^{\sqrt[d]{K}}_{\mathrm{ext}}$

Note how for $d = 1$, the commitments are the same as if logup* was used directly to implement the read-only memory argument. For larger memories, setting $d > 1$ significantly reduces the pushforward commitment cost from K to $\sqrt[d]{K}$ extension field elements.

5.2 Twist via logup*

Twist extends Shout to handle read/write memory. The prover commits to the following matrices and vectors:

- $\mathrm{Inc} : \mathbb{F}^{T \times K}$: The *sparse* matrix (at most T nonzero entries) indicating how each memory cell is incremented in each cycle.
- $\mathrm{wv} : \mathbb{F}^T$: The value written in each cycle.
- $\mathrm{ra}^{(i)} : \mathbb{F}^{T \times \sqrt[d]{K}}$ (for $1 \le i \le d$): *Sparse* matrices (T nonzero entries) indicating the read address, just like in Shout.
- $\mathrm{wa}^{(i)} : \mathbb{F}^{T \times \sqrt[d]{K}}$ (for $1 \le i \le d$): *Sparse* matrices (T nonzero entries); like $\mathrm{ra}^{(i)}$ but indicating the write address.

The protocol is best summarized in Figure 9 of [5]. As in Shout, $\mathrm{ra}^{(i)}$ and $\mathrm{wa}^{(i)}$ are matrices with one-hot rows. The commitment scheme from Section 4 is directly applicable.

For Inc, the scheme is not directly applicable: Even though it is also a matrix with at most one non-zero entry per row, the value of that entry could be any field element. However, for an honest prover, the *position* of that entry is the same as in the one-hot encoding of the write address. Therefore, instead of committing to $\mathrm{Inc} \in \mathbb{F}^{T \times K}$, the prover commits to a vector $\mathrm{Inc}_{\mathrm{val}} \in \mathbb{F}^T$ containing only the *value* of the increment. The following sum-check can be used to evaluate $\widetilde{\mathrm{Inc}}$:

$$\widetilde{\mathrm{Inc}}(r^{(1)}_{\mathrm{address}}, \ldots, r^{(d)}_{\mathrm{address}}, r_{\mathrm{cycle}}) = \sum_{j \in \{0,1\}^{\log(T)}} \widetilde{\mathrm{eq}}(j, r_{\mathrm{cycle}}) \cdot \widetilde{\mathrm{Inc}}_{\mathrm{val}}(j) \cdot \Big(\prod_{i=1}^{d} \widetilde{\mathrm{wa}}^{(i)}(r^{(i)}_{\mathrm{address}}, j)\Big) \tag{6}$$

Handling Inc by committing to a dense version, it becomes clear that wv could actually be virtual, saving the commitment cost: Similar to how Twist already uses virtual read values, the value of the written cell *before* the write can be obtained as a virtual polynomial via an analogous

read-checking sum-check using $\mathrm{wa}^{(i)}$. By adding $\mathrm{Inc}_{\mathrm{val}}$, we obtain a virtual polynomial $\widetilde{\mathrm{wv}}$. This optimization has been confirmed by the authors[3].

Using batching, we obtain a read/write memory argument with the following commitment costs:

- d read address vectors $\mathrm{ra}^{(i)}_{\mathrm{dense}} \in \mathbb{F}^T$.
- d write address vectors $\mathrm{wa}^{(i)}_{\mathrm{dense}} \in \mathbb{F}^T$.
- the increment vector $\mathrm{Inc}_{\mathrm{val}} \in \mathbb{F}^T$.
- one batched pushforward $P \in \mathbb{F}_{\mathrm{ext}}^{\sqrt[d]{K}}$ for the read and write addresses.

Note that in the context of zkVMs, K might be very small. For instance, for RISC-V registers, $K = 32$. In this case, even for $d = 1$, the commitment cost for the pushforwards is likely negligible.

6 Conclusion

We have shown how to apply techniques from logup* to implement Twist and Shout in hash-based proof systems. The main observation is that the one-hot commitment scheme implicit in logup* can be used to commit to the address matrices with one-hot rows required by Twist and Shout. This enables improvements in zkVMs built on top of hash-based multilinear commitment schemes:

- Shout provides a cheap indexed lookup argument. Compared to using logup* directly, it is also applicable for large tables.
- Twist provides a cheap read/write memory argument. Compared to techniques based on offline memory checking, commitment cost is significantly reduced.

Acknowledgements

We thank Jonathan Wang for prompting this work by suggesting that one-hot vectors could be looked up via logup* and for his insightful feedback. We also thank Justin Thaler for confirming the optimization regarding the write values in Twist, and Quang Dao for raising the question of batching. Finally, we thank Shuang Wu for helpful discussions and Leonardo Alt for proofreading.

References

[1] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2):225–244, 1994.

[2] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[3] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

[4] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Paper 2023/1284, 2023.

[3] A description of this optimization can be found at: `https://jolt.a16zcrypto.com/how/twist-shout.html#wv-virtualization`

[5] Srinath Setty and Justin Thaler. Twist and shout: Faster memory checking arguments via one-hot addressing and increments. Cryptology ePrint Archive, Paper 2025/105, 2025.

[6] Lev Soukhanov. Logup*: faster, cheaper logup argument for small-table indexed lookups. Cryptology ePrint Archive, Paper 2025/946, 2025.

[7] Justin Thaler et al. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022.