Math Companion to Soundcalc

December 9, 2025

Contents

1 Preliminaries

1.1 Fields

Fields of size q are denoted as $\mathbb{F}_q$ or simply $\mathbb{F}$.

1.2 Regime-specific bounds

1.2.1 Proximity parameter

`unique_decoding.py/get_proximity_parameter()`

In UDR

$$\delta_U(\rho) = (1-\rho)/2 \tag{1}$$

In JBR we compute the δ differently.

`johnson_bound.py/get_proximity_parameter()`

$$\delta_J(\rho, N, q) = \begin{cases} 1 - \sqrt{\rho} - \frac{1}{N} & \text{if } q > 2^{150} \\ 1 - \sqrt{\rho} - \frac{\rho}{20} & \text{otherwise} \end{cases} \tag{2}$$

1.2.2 List sizes

`unique_decoding.py/get_max_list_size()`

$$\ell = 1$$

`johnson_bound.py/get_max_list_size()`

We compute

$$\ell(\rho, N) = \frac{1}{2(1-\rho-\delta)\sqrt{\rho}}.$$

1.2.3 Batching errors

We define batching error functions

$$\epsilon_{\text{batch,pow},J}(\rho, N, q, B), \quad \epsilon_{\text{batch,pow},U}(\rho, N, q, B), \quad \epsilon_{\text{batch,lin},U}(\rho, N, q), \quad \epsilon_{\text{batch,pow},U}(\rho, N, q, B)$$

For JBR :

- Compute proximity parameter δ as in (2):
- Compute m as in `get_m()`:

$$m = 0.5 + \max\left(\left\lceil \frac{\sqrt{\rho}}{1-\sqrt{\rho}-\delta} \right\rceil, 3\right)$$

- Compute linear error[1]:

$$\epsilon_{\text{batch,lin},J}(\rho, N, q) = \frac{N\left(2m^5 + 3m\delta\rho\right) + 3m\rho}{3\rho^{1.5}q}$$

- Compute powers error:

$$\epsilon_{\text{batch,pow},J}(\rho, N, q, B) = \epsilon_{\text{batch,lin},J}(\rho, N, q) \cdot (B-1)$$

For UDR :

- Compute linear error:

$$\epsilon_{\text{batch,lin},U}(\rho, N, q) = \frac{N}{q\rho}$$

- Compute powers error:

$$\epsilon_{\text{batch,pow},U}(\rho, N, q, B) = \epsilon_{\text{batch,lin},J}(\rho, N, q) \cdot B$$

[1] Code refers to Theorem 4.2 from BCHKS25

2 WHIR-based VM security level calculation

2.1 Notation and parameters

- Number of iterations: $M = \texttt{num\_iterations}$
- Iteration index: $i \in \{0, \dots, M-1\}$
- Folding round index: $s \in \{0, \dots, k-1\}$
- Field size: $|\mathbb{F}| = q$
- Constraint degree: $d = \texttt{constraint\_degree}$
- Folding factor: $k = \texttt{folding\_factor}$
- Log-degree in iteration i: $m_i = \texttt{log\_degrees}[i]$, and we set $m_i = m_0 - k \cdot i$
- Log-inverted rate in iteration i: $\mu_i = \texttt{log\_inv\_rates}[i]$, and we set $\mu_i = \log_2 \frac{1}{\rho} + (k-1) \cdot i$
- Iteration-round-specific RS codes:

$$\mathcal{C}_{i,s} = (\mathcal{C}_{i,s}[\rho], \mathcal{C}_{i,s}[N])_{i,s} = (2^{-\mu_i}, 2^{m_i - s})$$

- Number of OOD samples in iteration i: $w_i = \texttt{num\_ood\_samples}[i]$
- Number of queries in iteration i: $t_i = \texttt{num\_queries}[i]$
- Grinding bits:

$$g_{\text{batch}}, \quad \{g_{i,s}^{\text{fold}}\}_{i \in [M], s \in [k]}, \quad \{g_i^{\text{ood}}\}_{i \in M}, \quad \{g_i^{\text{qry}}\}_{i \in M}.$$

2.2 Bits of Security in UDR and JBR

In order to compute the bits of security in UDR and JBR regimes, we compute the following error terms using the parameters from section 2.1.

Computed in `get_security_levels_for_regime()`.

For any error term:

$$\lambda(\epsilon) = \lfloor -\log_2 \epsilon \rfloor.$$

Overall security:

$$\lambda_{\text{total}} = \min\{\lambda_{\text{batch}}, \{\lambda_{i,s}^{\text{fold}}\}_{i \in [M], s \in [k]}, \{\lambda_i^{\text{out}}\}_{i \in \{1, \dots, M-1\}}, \{\lambda_i^{\text{shift}}\}_{i \in \{1, \dots, M-1\}}, \lambda^{\text{fin}}\}.$$

2.3 Batching Error ϵ_{batch}

This computes the batching error with parameters from section 2.1. `get_batching_error()`

In the Johnson Bound Regime (JBR) we proceed as follows:

- Set code parameters

$$(\rho, N) = \left(2^{-\mu_0}, 2^{m_0}\right)$$

- Compute base error (cf. section 1.2.3):

$$\varepsilon_{\text{batch},JBR}^{\text{base}} = \begin{cases} \epsilon_{\text{batch,pow},J}(\rho, N, q, B) & \text{(power batching)}, \\ \epsilon_{\text{batch,lin},J}(\rho, N, q) & \text{(linear batching)}. \end{cases}$$

Whereas in the UDR we do as follows:

- Compute base error (cf. section 1.2.3):

$$\varepsilon_{\text{batch},UDR}^{\text{base}} = \begin{cases} \epsilon_{\text{batch,pow},U} & \text{(power batching)}, \\ \epsilon_{\text{batch,lin},U} & \text{(linear batching)}. \end{cases}$$

In both regimes we do as follows after grinding:

$$\epsilon_{\text{batch}} = \varepsilon_{\text{batch}}^{\text{base}} \cdot 2^{-g_{\text{batch}}}.$$

2.4 Folding Error

This section computes the error of a folding round as in `whir_based_vm.py/epsilon_fold()`, referring to Theorem 5.2 of the WHIR paper.

For iteration $i \in [M]$ and folding round $s \in [k]$ in JBR we do[2]:

- Get list size as in section 1.2.2
$$\ell_{i,s} = \ell(\mathcal{C}_{i,s}[\rho])$$
- Base error (two terms):
$$\varepsilon_{i,s}^{\text{fold,base,J}} = d \cdot \frac{\ell_{i,s}}{q} + \epsilon_{\text{batch,pow},J}(\mathcal{C}_{i,s+1}[\rho], \mathcal{C}_{i,s+1}[N], q, B = 2).$$

And in UDR the base error is
$$\varepsilon_{i,s}^{\text{fold,base,U}} = \frac{d}{q} + \epsilon_{\text{batch,pow},U}(\mathcal{C}_{i,s+1}[\rho], \mathcal{C}_{i,s+1}[N], q, B = 2).$$

After grinding:
$$\epsilon_{i,s}^{\text{fold}} = \varepsilon_{i,s}^{\text{fold,base}} \cdot 2^{-g_{i,s}^{\text{fold}}}.$$

2.5 OOD error

This computes the Out-of-Domain error as in `whir_based_vm.py/epsilon_out()`.

For iteration $i \in \{1, 2, \dots, M-1\}$

- Base error in JBR (cf. section 1.2.2):
$$\varepsilon_i^{\text{out,base,J}} = \ell_{i,0}^2 \left(\frac{2^{m_i}}{2q}\right)^{w_{i-1}}.$$
- Base error in UDR:
$$\varepsilon_i^{\text{out,base,U}} = \left(\frac{2^{m_i}}{2q}\right)^{w_{i-1}}.$$
- After grinding:
$$\epsilon_i^{\text{out}} = \varepsilon_i^{\text{out,base}} \cdot 2^{-g_{i-1}^{\text{ood}}}.$$

2.6 Shift Error

`epsilon_shift()`

For iteration $i \in \{1, 2, \dots, M-1\}$ in JBR

- Get δ for the iteration using (2):
$$\delta_i = \min_{s \in [k+1]} \delta_J(\mathcal{C}_{i-1,s}[\rho], \mathcal{C}_{i-1,s}[N], q)$$
- Base error (two terms):
$$\varepsilon_i^{\text{shift,base}} = (1 - \delta_i)^{t_{i-1}} + \ell_{i,0} \cdot \frac{t_{i-1} + 1}{q}.$$

Same procedure in UDR:

- Get δ for the iteration using (1):
$$\delta_i = \min_{s \in [k+1]} \delta_U(\mathcal{C}_{i-1,s}[\rho])$$
- Base error :
$$\varepsilon_i^{\text{shift,base}} = (1 - \delta_i)^{t_{i-1}} + \frac{t_{i-1} + 1}{q}.$$

After grinding:
$$\epsilon_i^{\text{shift}} = \varepsilon_i^{\text{shift,base}} \cdot 2^{-g_{i-1}^{\text{qry}}}.$$

[2] The function `epsilon_fold()` counts folding rounds from 1 to k, whereas here we count from 0 to $k-1$.

2.7 Final Round Error

`epsilon_final()`
JBR:

- Get δ for the iteration using (2):
$$\delta_{M-1} = \min_{s\in[k+1]} \delta_J(\mathcal{C}_{M-1,s}[\rho])$$

UDR:

- Get δ for the iteration using (1):
$$\delta_{M-1} = \min_{s\in[k+1]} \delta_U(\mathcal{C}_{M-1,s}[\rho])$$

Then the base error is :
$$\varepsilon^{\text{fin,base}} = (1-\delta_{M-1})^{t_{M-1}}.$$

After grinding:
$$\epsilon^{\text{fin}} = (1-\delta_{M-1})^{t_{M-1}} \cdot 2^{-g^{\text{qry}}_{M-1}}.$$

3 WHIR Proof Size Calculations

This section summarizes the proof-size formula computed in `get_proof_size_bits()`.

3.1 Initial Function Size

$$S_{\text{if}} = b_{\text{hash}}.$$

3.2 Initial Sumcheck Size

Thus the folding proof size per round is
$$S_{\text{sumcheck}} = k(d-1)q.$$

3.3 OOD Proof Size for Iteration i

$$S_{\text{ood},i} = b_{\text{hash}} + w_i q + k(d-1)q$$

$$S_{\text{ood},M} = 2^{m_M} q$$

3.4 Query Proof Size for Iteration i

Thus
$$S_{\text{qry},i} = \begin{cases} t_i \cdot MP(2^{m_i+\mu_i-k}, 2^k, \log_2 q, b_{\text{hash}}). & i > 0 \\ t_i \cdot MP(2^{m_0+\mu_0-k}, 2^k, \log_2 p, b_{\text{hash}}). & M = 1 \end{cases}$$

where p is characteristic of the field $\mathbb{F}_q$.

3.5 Total Proof Size

Collecting all terms and summing over all M iterations:

$$\boxed{S_{\text{total}} = S_{\text{if}} + S_{\text{sumcheck}} + \sum_{i\in[M]} (S_{\text{ood},i} + S_{\text{qry},i})}$$

4 FRI-based VM security level calculation

This section calculates the security level for a FRI-based VM in section 4.2.

4.1 FRI parameters

Global parameters used in the FRI analysis:

- r_{FRI} — number of FRI rounds.
- Folding factors $\widehat{\text{folds}} = [k_0, k_1, \ldots, k_{r_{FRI}-1}]$;
- t — number of queries.
- δ — proximity parameter.
- ρ — rate of the Reed-Solomon code.
- N — trace length.
- ℓ — list size.
- $b_{\text{grind},Q}$ — grinding parameter for the query phase.
- n — witness size.
- b_{hash} — number of bits in the hash function output.
- b_{proof} — proof size in bits.
- B — batch size.

4.2 Security level for a FRI-based VM

The security level is calculated in

`zkvms/fri_based_vm.py/get_security_levels()`

.

It is done separately for two different regimes: UDR and JBR — using the same procedure `fri_based_vm.py/get_security_l`

1. Calculate the FRI round-by-round soundness errors $\mathbf{e}_{\text{FRI}}$:
$$\mathbf{e}_{\text{FRI}} = \max(\text{e}_{\text{batch}}, \{\epsilon_i^{\text{fold}}\}_{i\in[r_{FRI}]}, \text{e}_{\text{query}})$$
2. Obtain optimal δ proximity parameter as by section 1.2.1.
3. Obtain the list size ℓ for the respective δ as by section 1.2.2.
4. Obtain the DEEP-ALI soundness error ϵ_{DA} as by section 4.2.1.
5. Compute the total soundness error as
$$\epsilon = \max(\mathbf{e}_{\text{FRI}}, \epsilon_{\text{DA}}).$$

Then the full security level in bits is the maximum of the two regimes:

$$\text{Security level} = \max(-\log_2 \epsilon_U, -\log_2 \epsilon_J).$$

4.2.1 DEEP-ALI errors

`fri_based_vm.py/get_DEEP_ALI_errors()`

The DEEP-ALI soundness error is computed as follows:

$$\epsilon_{\text{DA}} = \max\left(\frac{L^+ \cdot C}{q}, \frac{L^+(d(N + m_c - 1) + (N - 1))}{q - N - D}\right) \tag{3}$$

4.3 Batching Error

This computes the batching error with parameters from section 2.1. `get_batching_error()`

In the Johnson Bound Regime (JBR) we compute base error as:

$$e_{\text{batch}}^{\text{base,JBR}} = \begin{cases} \epsilon_{\text{batch,pow},J}(\rho, N, q, B) & \text{(power batching)}, \\ \epsilon_{\text{batch,lin},J}(\rho, N, q) & \text{(linear batching)}. \end{cases}$$

Whereas in the UDR we compute base error:

$$e_{\text{batch}}^{\text{base,UDR}} = \begin{cases} \epsilon_{\text{batch,pow},U}(\rho, N, q, B) & \text{(power batching)}, \\ \epsilon_{\text{batch,lin},U}(\rho, N, q) & \text{(linear batching)}. \end{cases}$$

4.4 Folding errors

This is calculated in `fri_based_vm.py/get_commit_phase_error()`.

For round $i \in [r_{FRI}]$ we compute the dimension N_i as

$$N_i = \frac{N}{\prod_{j \in [i]} k_j}$$

Then the folding error in round i is computed as

$$\epsilon_i^{\text{fold}} = \begin{cases} \epsilon_{\text{batch,pow},J}(\rho, N_i, q, k_i), & \text{JBR}, \\ \epsilon_{\text{batch,pow},U}(\rho, N_i, q, k_i), & \text{UDR} \end{cases} \tag{4}$$

4.5 Query error

This is calculated in `fri_based_vm.py/get_query_phase_error()`.

Query phase error:

$$\epsilon_{\text{query}} = (1 - \theta)^t \cdot 2^{-b_{\text{grind},Q}} \tag{5}$$

5 FRI proof size

This calculation is performed in

`fri.py/get_FRI_proof_size_bits()`

. The FRI proof contains two parts: Merkle roots, and one "openings" per query, where an "opening" is a Merkle path for each folding layer. For each layer we count the size that this layer contributes, which includes the root and all Merkle paths.

Initial round: one root and one path per query. We assume that for the initial functions, there is only one Merkle root, and each leaf i for that root contains symbols i for all initial functions.

Folding rounds: we assume that "siblings" for the following layers are grouped together in one leaf. This is natural as they always need to be opened together.

The proof size is calculated as follows:

$$\begin{aligned} b_{\text{proof}} = & \underbrace{b_{\text{hash}} + t \cdot MP(\frac{n}{\widehat{\text{folds}}[0]}, B, \log_2 |\mathbb{F}|, b_{\text{hash}})}_{\text{Initial round}} + \\ & + \underbrace{\sum_{1 \le i \le r_{FRI}-2} \left(b_{\text{hash}} + t \cdot MP(\frac{n}{\prod_{1 \le j \le i} \widehat{\text{folds}}[j]}, B, \log_2 |\mathbb{F}|, b_{\text{hash}}) \right)}_{\text{Folding rounds but last}} + \\ & + \underbrace{\left(b_{\text{hash}} + t \cdot MP(\frac{n}{\widehat{\text{folds}}[r_{FRI}-1] \prod_{1 \le j \le r_{FRI}-1} \widehat{\text{folds}}[j]}, B, \log_2 |\mathbb{F}|, b_{\text{hash}}) \right)}_{\text{Last folding round}} \end{aligned} \tag{6}$$

where $MP(n, s, q, b)$ is the Merkle path size calculated as

$$MP(n, s, q, b) = \underbrace{sq}_{\text{leaf size}} + \underbrace{sq}_{\text{sibling}} + \underbrace{\lceil \log_2 n \rceil \cdot b}_{\text{co-path}} \tag{7}$$