HIT: Single-Pass Inference for Single-Trace NTT Side-Channel Attacks

Yihan Nie, Yuxuan Wang, Yubo Zhao, Zixin He, Wenjing Zheng
Jintong Yu, Shifeng Sun, Chi Zhang, Pei Cao*, Dawu Gu*
Shanghai Jiao Tong University *Corresponding authors: loccs_cp@sjtu.edu.cn, dwgu@sjtu.edu.cn

***Abstract*—In physical implementations of the post-quantum standards ML-KEM and ML-DSA, the Number-Theoretic Transform (NTT) directly processes secret polynomials and is a primary target for single-trace side-channel key recovery. Existing Soft Analytical Side-Channel Attacks (SASCA) apply loopy Belief Propagation (BP) over cyclic $\mathbb{Z}_q$ factor graphs, where BP fails under high noise and $O(q^2)$ updates make ML-DSA infeasible. We present the Hierarchical Inference Tree (HIT), a framework built on the insight that small secret inputs deterministically fix all NTT intermediates across the recursive butterfly network. Instead of loopy BP, HIT merges and prunes candidate inputs from leaves to root in a single pass on an acyclic binary tree. This approach eliminates belief oscillation, guarantees a constant runtime, and decouples complexity from q. On ML-KEM-768, HIT runs over $900\times$ faster than state-of-the-art SASCA and more than doubles noise tolerance. HIT also enables the first single-trace analytical attack on the ML-DSA NTT and recovers each polynomial in under one second. On physical ARM Cortex-M4 targets, HIT achieves over 98% polynomial recovery across both standards with only 20 to 80 profiling traces.**

1. Introduction

With the NIST standardization of ML-KEM (FIPS 203) and ML-DSA (FIPS 204) [1], [2], lattice-based post-quantum cryptography has moved toward real-world deployment on embedded devices. On these platforms, physical side-channel attacks exploit power consumption and Electromagnetic (EM) emissions to extract secret keys directly during cryptographic executions [3], [4]. Side-channel studies on lattice implementations have targeted operations such as polynomial multiplication [5], [6], [7], decapsulation with chosen ciphertexts [3], [8], [9], and secret key decode steps [10]. Among these operations, the Number-Theoretic Transform (NTT) serves as a primary target for key recovery [11], [12], [13]. In both standards, the NTT accelerates polynomial multiplication over $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ and directly processes the polynomials of the secret vector $\mathbf{s}$ in ML-KEM and $\mathbf{s}_1$ in ML-DSA [1], [2]. Exact recovery of these NTT inputs exposes the ML-KEM decapsulation key or enables signature forgery on arbitrary messages in ML-DSA [14], [15]. Many prior attacks require multiple attack traces [4], [5], [16] or adaptive chosen-ciphertext queries [8]. By contrast, a single-trace attack on the forward NTT [12], [13] can recover a secret polynomial from a single attack trace of ML-KEM key generation or ML-DSA signing and thus poses a severe practical threat.

Yet in a single noisy trace, an adversary cannot simply guess each of the 256 coefficients independently, since even a small error probability per coefficient makes full polynomial recovery nearly impossible [10]. Instead, Soft Analytical Side-Channel Attacks (SASCA) [17] exploit the algebraic constraints across the butterfly layers of the NTT on a bipartite factor graph [11], [12]. In this graph, variable nodes represent the intermediate values in $\mathbb{Z}_q$ and factor nodes represent the butterfly operations and side-channel leakages. Belief Propagation (BP) then passes messages between these nodes to compute the marginal distributions of the secret coefficients. Subsequent studies adapt the factor graph for lazy reduction [18], omit early NTT layers [19], or train deep neural networks to supply calibrated leakage likelihoods (Neur-SASCA [13]).

Despite these refinements, all existing NTT SASCA methods still apply BP over a cyclic $\mathbb{Z}_q$ factor graph and share two fundamental limitations. First, graph cycles severely limit noise tolerance. While standard BP is exact on acyclic trees, butterfly connections across NTT layers form dense cycles that force BP to pass messages iteratively around the loops as loopy BP [20]. Without convergence guarantees, loopy BP often oscillates or converges to incorrect marginals under high noise [20], [21]. Second, $\mathbb{Z}_q$ inference makes large moduli intractable. Specifically, message updates incur an $O(q^2)$ cost per factor node, which grows from $q^2 \approx 1.1\times10^7$ operations in ML-KEM ($q = 3{,}329$) to a prohibitive $q^2 \approx 7.0\times10^{13}$ in ML-DSA ($q = 8{,}380{,}417$) and increases even further under lazy reduction [12], [13], [18].

Prior refinements cannot overcome either limitation because they remain tied to BP, which propagates $\mathbb{Z}_q$ distributions over intermediate butterfly states. We depart from this paradigm and recast single-trace key recovery directly in the secret input domain. The secret inputs deterministically fix every intermediate integer throughout the transform, yet their support $[-\eta, \eta]$ contains only five to nine values per coefficient and is three to six orders of magnitude smaller than $\mathbb{Z}_q$. Moreover, whereas intermediate variables form dense cycles across layers, the radix-2 butterflies recursively pair disjoint subsets of secret inputs into an acyclic binary tree of nested regions. In this representation, both graph cycles and $\mathbb{Z}_q$ message updates disappear by construction.

Yet over all 256 secret coefficients, the joint input space grows exponentially and rules out exhaustive enumeration. To make inference in the secret input domain tractable, we propose the **Hierarchical Inference Tree (HIT)**. Along

the binary region tree, HIT merges disjoint input subsets and prunes unlikely candidates from leaves to root in a single pass. This traversal eliminates belief oscillation, guarantees a constant runtime across noise levels, and preserves high accuracy under severe noise. At each node, HIT evaluates exact signed integers under lazy reduction and scores the combined input assignments rather than all $\mathbb{Z}_q$ states. Consequently, HIT decouples complexity from q and makes ML-DSA practical from a single trace.

Contributions. We summarize our main contributions in three parts.

- **A new analytical paradigm for single-trace NTT attacks (Section 3).** We propose HIT to replace loopy BP on cyclic $\mathbb{Z}_q$ factor graphs with a single bottom-up pass over an acyclic binary tree of NTT regions. At each region, HIT evaluates exact integer boundary vectors and retains at most K candidates to bound the per-node merge cost to $O(K^2)$ instead of $O(q^2)$. This approach eliminates belief oscillation, guarantees a constant runtime across noise levels, and decouples the computational complexity from the modulus q.
- **Inference efficiency and noise tolerance on ML-KEM (Sections 4.2 and 4.4).** On ML-KEM-768, HIT accelerates inference by over $900\times$ compared with state-of-the-art SASCA methods [12], [13] and more than doubles the tolerable noise threshold. On physical ARM Cortex-M4 power traces, HIT achieves 100.0% exact recovery of both individual polynomials and the complete secret vector $\mathbf{s}$ with only 20 profiling traces ($8\times$ fewer than [13]). It also maintains high accuracy across four physical devices, five compiler optimization levels (`-O0` to `-O3` and `-Os`), and EM measurements with a low signal-to-noise ratio (SNR).
- **First single-trace analytical attack on the ML-DSA NTT (Section 4.3).** HIT overcomes the $O(q^2)$ modulus barrier ($q = 8{,}380{,}417$) to make single-trace analytical attacks on the ML-DSA NTT practical, and recovers each secret polynomial in under 1 s on an NVIDIA RTX 4090. On physical Cortex-M4 power traces, HIT achieves 100.0% exact polynomial recovery for ML-DSA-44 and 98.7% for ML-DSA-65 with 80 profiling traces, and recovers the complete ML-DSA-44 secret vector $\mathbf{s}_1$ at 100.0% with 20 profiling vectors. Furthermore, the attack generalizes across physical devices and EM measurements.

The code is available online.[1]

2. Background

Notation. We denote the polynomial ring $\mathbb{Z}_q[X]/(X^{256}+1)$ by R_q and its NTT-domain representation by T_q. Bold lowercase and uppercase letters denote vectors and matrices over R_q or T_q, respectively. We write $\hat{f} = \mathrm{NTT}(f)$ for the NTT of a polynomial f (applied component-wise to vectors and matrices) and use $\circ$ to denote coefficient-wise multiplication in T_q.

1. https://anonymous.4open.science/r/HIT-B93B/

Algorithm 1 ML-KEM key generation: simplified K-PKE core [1]

Require: Random seed d
1: Derive seeds ρ, σ from d and parameter k
2: Expand $\hat{\mathbf{A}}$ from ρ. Sample $\mathbf{s}, \mathbf{e}$ from CBD_{η_1} using σ
3: $\hat{\mathbf{s}} \leftarrow \mathrm{NTT}(\mathbf{s})$ ▷ target
4: $\hat{\mathbf{e}} \leftarrow \mathrm{NTT}(\mathbf{e})$
5: $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
6: **return** encoded $(\hat{\mathbf{t}}, \rho)$ and $\hat{\mathbf{s}}$ as the K-PKE keys

2.1. ML-KEM and ML-DSA

ML-KEM. ML-KEM [1], [22] enables two parties to establish a shared key through encapsulation and decapsulation. During key generation, it samples a secret vector $\mathbf{s} \in R_q^k$ whose coefficients follow a centered binomial distribution CBD_{η_1} over $[-\eta_1, \eta_1]$. It then computes $\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s})$ and forms the public-key component $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$, where $\hat{\mathbf{A}}$ is a public matrix and $\mathbf{e}$ is a sampled error vector. Our attack targets the forward NTT of $\mathbf{s}$ during key generation (Algorithm 1, Line 3), which executes only once per keypair to compute the secret component $\hat{\mathbf{s}}$ of the decapsulation key.
ML-DSA. ML-DSA [2], [23] generates digital signatures from a secret vector $\mathbf{s}_1 \in R_q^\ell$ whose coefficients are uniformly distributed over $[-\eta, \eta]$. During signing, ML-DSA computes the signature component $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ from an ephemeral masking vector $\mathbf{y}$ and a sparse challenge polynomial c, and evaluates the polynomial product $c\mathbf{s}_1$ via the NTT. Inside the signing loop, operations such as mask expansion (Line 4) [4], commitment rounding (Line 6) [16], challenge multiplication $c\mathbf{s}_1$ (Line 7) [5], [24], and rejection checks (Line 8) [25] interact with varying ephemeral masks or challenges across multiple signatures. By contrast, our attack targets the forward NTT of $\mathbf{s}_1$ (Algorithm 2, Line 1), which operates solely on the fixed secret $\mathbf{s}_1$. As shown by Groot Bruinderink and Pessl [14] and Ravi et al. [15], an adversary who recovers $\mathbf{s}_1$ can combine it with the public key to forge valid signatures on arbitrary messages without $\mathbf{s}_2$ or $\mathbf{t}_0$.

Table 1 summarizes the parameters of both standards and contrasts the large ring modulus q with the small secret coefficient bound (η_1 in ML-KEM and η in ML-DSA, both denoted simply by $\eta \in \{2, 3, 4\}$ hereafter).

2.2. Number-Theoretic Transform

The NTT accelerates polynomial multiplication in R_q by transforming a 256-coefficient polynomial $\mathbf{x} = (x_0, \ldots, x_{255})$ into T_q across D successive layers. ML-KEM uses an incomplete NTT of $D = 7$ layers that splits $\mathbf{x}$ into two independent 128-coefficient subtransforms,

Algorithm 2 ML-DSA signing: simplified arithmetic core [2]

Require: Decoded key components, message representative μ, sampling seed ρ''

$\hat{\mathbf{s}}_1 \leftarrow \mathrm{NTT}(\mathbf{s}_1)$ ▷ target

Expand $\hat{\mathbf{A}}$. Precompute the transforms of $\mathbf{s}_2, \mathbf{t}_0$. Set $\kappa \leftarrow 0$

while no signature has been accepted **do**

  $\mathbf{y} \leftarrow \mathrm{ExpandMask}(\rho'', \kappa)$. Set $\kappa \leftarrow \kappa + \ell$

  $\mathbf{w} \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{y}))$

  Derive $\tilde{c}$ from $\mu, \mathrm{HighBits}(\mathbf{w})$. Sample c from $\tilde{c}$

  $\mathbf{z} \leftarrow \mathbf{y} + \mathrm{NTT}^{-1}(\mathrm{NTT}(c) \circ \hat{\mathbf{s}}_1)$

  Compute hints $\mathbf{h}$ and apply the standard rejection checks

return the encoding of $(\tilde{c}, \mathbf{z} \bmod^{\pm} q, \mathbf{h})$

TABLE 1. SELECTED STANDARD PARAMETERS [1], [2]. HERE k IS THE LENGTH OF $\mathbf{s}$ IN ML-KEM AND $\mathbf{s}_2$ IN ML-DSA. THE PARAMETER ℓ IS THE LENGTH OF ML-DSA'S $\mathbf{s}_1$. THE PARAMETERS η_1 AND η ARE THE SECRET COEFFICIENT BOUNDS.

Parameter set	q	k	ℓ	η_1/η
ML-KEM-512	3329	2	–	3
ML-KEM-768	3329	3	–	2
ML-KEM-1024	3329	4	–	2
ML-DSA-44	8380417	4	4	2
ML-DSA-65	8380417	6	5	4
ML-DSA-87	8380417	8	7	2

whereas ML-DSA uses a complete NTT of $D = 8$ layers that couples all 256 coefficients.

Each NTT layer consists of 128 radix-2 butterfly operations, indexed by layer $t \in \{0, \ldots, D-1\}$. A butterfly combines two inputs a and b with a known twiddle factor ζ to produce

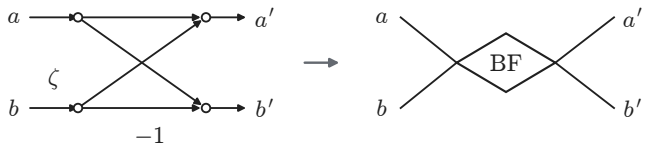
$$a' \equiv a + \zeta b \pmod{q}, \qquad b' \equiv a - \zeta b \pmod{q}. \quad (1)$$

Figure 1 illustrates this computation and its abstraction as a butterfly (BF) node.

Lazy reduction accelerates NTT computation by omitting modular reductions after selected operations [12]. In the PQClean implementations we target, butterfly additions and subtractions are left unreduced, while multiplications use Montgomery reduction [26]. These implementations store the values at each NTT layer as signed 16-bit integers in ML-KEM and signed 32-bit integers in ML-DSA. Integers that are congruent modulo q can have different binary representations and Hamming weights. The leakage likelihood therefore depends on the raw integer representation, not just its residue modulo q.

2.3. Side-Channel Analysis and Profiling

Side-channel analysis exploits physical leakage from cryptographic implementations to recover secret information [27]. In profiled attacks, the adversary first collects N profiling traces from a controlled device to build a leakage model, and then applies this model to a single target trace to infer the secret input polynomial [28]. Profiling and attack may take place either on the same physical device (*same-device* setting) or across different devices (*cross-device* setting) [29].



(a) Butterfly (b) BF representation

Figure 1. A forward NTT butterfly and its abstraction as a BF node. Panel (a) shows the signal-flow representation. Edge labels denote multiplicative weights, and incoming values are added at each output. Panel (b) shows the corresponding BF node with the same inputs a, b and outputs a', b'.

Template attacks (TA) model the leakage of intermediate values with multivariate Gaussian distributions [28], [30]. We group the N profiling traces by the Hamming weight (HW) h of the corresponding intermediate value and fit a Gaussian template $\ell^{\mathrm{HW}}(L \mid h)$ for each group. Given leakage observations L at selected points of interest (PoIs), the likelihood of a candidate integer z is

$$\ell(L \mid z) = \ell^{\mathrm{HW}}(L \mid \mathrm{HW}_w(z)), \quad (2)$$

where $w \in \{16, 32\}$ denotes the integer bit width. Because multiple values share the same Hamming weight and are indistinguishable from a single leakage observation, secret recovery must combine leakages across multiple NTT layers.

2.4. SASCA and Belief Propagation

Factor-graph representation. SASCA combines side-channel leakage likelihoods with the NTT's butterfly constraints to recover the secret input polynomial from a single trace [11], [12]. It represents the NTT as a bipartite factor graph in which each butterfly is a factor node connected to variable nodes for its inputs and outputs [31]. The butterfly factor encodes the algebraic constraint as

$$\psi_{\mathrm{BF}}(a, b, a', b') = \begin{cases} 1, & a' \equiv a + \zeta b \pmod{q}, \\ & b' \equiv a - \zeta b \pmod{q}, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

Each leakage observation L is represented by a unary factor $\psi_{\mathrm{leak}}(z) = \ell(L \mid z)$ connected to the corresponding variable node, and each input coefficient is assigned a prior factor $p(x_i)$. Figure 2 illustrates this representation for a two-layer, four-input NTT region.

Belief propagation. BP estimates the marginal distribution of each NTT variable by passing messages between variable and factor nodes [31]. Let v_i denote a graph variable, ψ a factor, and $\mathcal{N}(\cdot)$ the set of neighboring nodes. The variable-to-factor and factor-to-variable messages are updated as

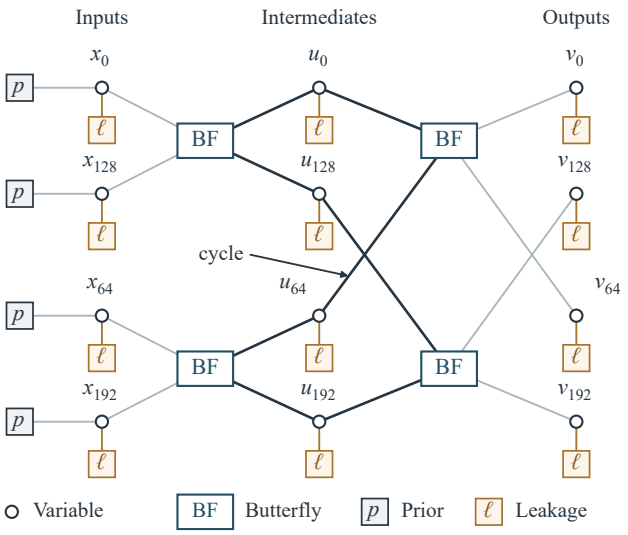


Figure 2. Factor graph of a two-layer, four-input NTT region. Circles represent NTT values, and BF boxes enforce butterfly constraints. Unary factors p and ℓ encode input priors and leakage likelihoods. Variables u_i and v_i denote values after the first and second layers. Dark edges highlight a cycle.

$$
\begin{aligned}
m_{i\to\psi}(v_i) &\propto \prod_{\phi\in\mathcal{N}(i)\setminus\{\psi\}} m_{\phi\to i}(v_i),\\
m_{\psi\to i}(v_i) &\propto \sum_{\mathbf{v}_{\mathcal{N}(\psi)\setminus\{i\}}} \psi(\mathbf{v}_{\mathcal{N}(\psi)}) \prod_{j\in\mathcal{N}(\psi)\setminus\{i\}} m_{j\to\psi}(v_j),
\end{aligned}
\tag{4}
$$

where each outgoing message excludes the incoming message from its recipient. The estimated marginal distribution of v_i is then

$$
\widehat{p}_i(v_i) = \frac{1}{Z_i} \prod_{\psi\in\mathcal{N}(i)} m_{\psi\to i}(v_i), \tag{5}
$$

where Z_i is a normalization constant, and v_i is estimated as the value with the highest marginal probability.

Limitations of loopy BP. While BP computes exact marginals on acyclic graphs, NTT factor graphs contain numerous cycles (Figure 2). Consequently, inference relies on iterative message updates known as *loopy BP*, which guarantee neither convergence nor exact marginals [12], [20], [21]. Figure 3 illustrates both failure modes on a small cyclic XOR factor graph whose constraints ($X \oplus Y = Z$, $X \oplus W = Z$, and $Y \oplus W = Z$) jointly imply $Z = 0$. Under different local likelihoods, the loopy BP estimate of $P(Z = 1)$ either oscillates indefinitely (Figure 3b) or converges to an incorrect marginal above $1/2$ that favors the impossible value $Z = 1$ (Figure 3c). In addition, each butterfly message update in Equation (4) over $\mathbb{Z}_q$ requires $O(q^2)$ operations per node [12]. This per-node cost increases from $q^2 \approx 1.1\times10^7$ in ML-KEM to a prohibitive $q^2 \approx 7.0\times10^{13}$ across each of the 1,024 butterfly nodes in ML-DSA [12], [13], and grows even further when variable domains expand beyond $\mathbb{Z}_q$ to track unreduced integers under lazy reduction [18].

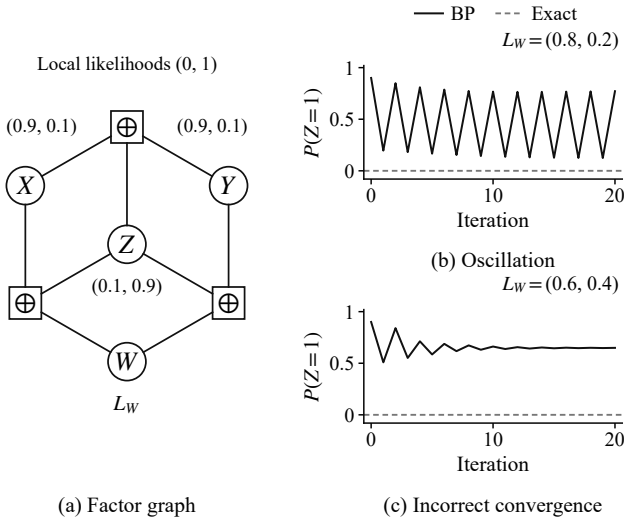


Figure 3. Two failure modes of loopy BP. (a) A cyclic XOR factor graph whose constraints jointly imply $Z = 0$. Local likelihoods are given in the order $(0, 1)$. The two settings of L_W produce (b) oscillation and (c) convergence to an incorrect marginal. Solid and dashed lines show the BP estimate and exact value of $P(Z = 1)$, respectively.

Prior NTT SASCA work. Primas et al. [11] first applied SASCA to the NTT for single-trace key recovery in lattice-based encryption, and PP19 [12] subsequently improved noise tolerance through a refined NTT factor graph and BP schedule. Subsequent studies extended PP19 in several directions. Assael et al. [18] adapted the factor graph for coefficient packing and lazy reduction, GRALE [19] omitted early NTT layers to reduce BP complexity, Neur-SASCA [13] trained neural networks with fingerprint regularization to supply calibrated leakage probabilities, and further works modified the factor graph to attack masked implementations [32], [33] or shuffled NTTs [34]. Although Wedenig et al. [35] bypassed loopy BP on the AES MixColumns operation by compiling its small state into a tractable probabilistic circuit, circuit compilation of modulo-q NTT butterflies remains infeasible for large q. Consequently, all existing NTT SASCA methods retain iterative BP over $\mathbb{Z}_q$ and remain limited by cycle-induced instability and the $O(q^2)$ modulus barrier.

3. HIT Framework

Figure 4 gives an overview of the complete HIT inference framework, which recasts single-trace recovery of the NTT secret inputs as a single bottom-up pass over a binary tree of NTT regions. Given input priors and local leakage likelihoods from an attack trace $\mathbf{L}$, HIT merges child candidates (Section 3.1), computes their combined weights from leakage in the joining layer (Section 3.2), and applies top-K pruning at internal depths (Section 3.3). At the target depth, it computes coefficient-wise estimates from all generated candidates. Section 3.4 extends HIT to masked NTT implementations.

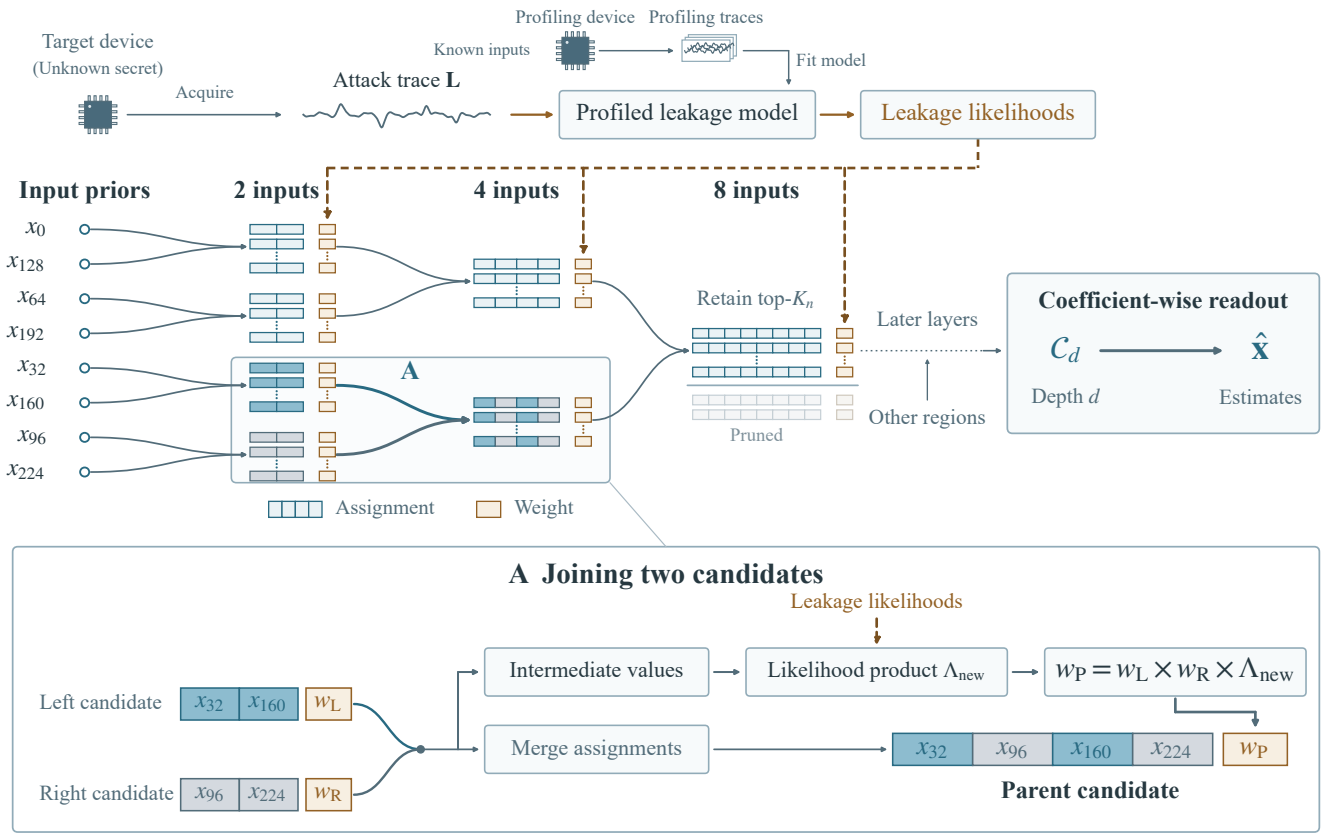


Figure 4. Overview of HIT inference. The target device supplies a single attack trace; a profiling device supplies traces with known inputs to fit the leakage model. The model converts the attack trace into local likelihoods, which feed updates to candidate weights through dashed arrows. Rows represent joint input assignments and unnormalized weights; vertical dots indicate omitted candidates. In the eight-input example, the horizontal line separates retained top-K_n candidates from faded, pruned candidates. Blue and gray cells identify the child regions in merge A, expanded below for one candidate pair. In the joining layer, their assignments determine the intermediate values and likelihood product Λ_{new} (Λ_{n+1} in Equation (8)), yielding $w_{\text{P}} = w_{\text{L}} w_{\text{R}} \Lambda_{\text{new}}$. Parent assignments follow the original input index order. At depth d, HIT sums the weights in $\mathcal{C}_d$ for each coefficient value and selects the largest marginal to form $\hat{\mathbf{x}}$. Waveforms and displayed candidate counts are illustrative.

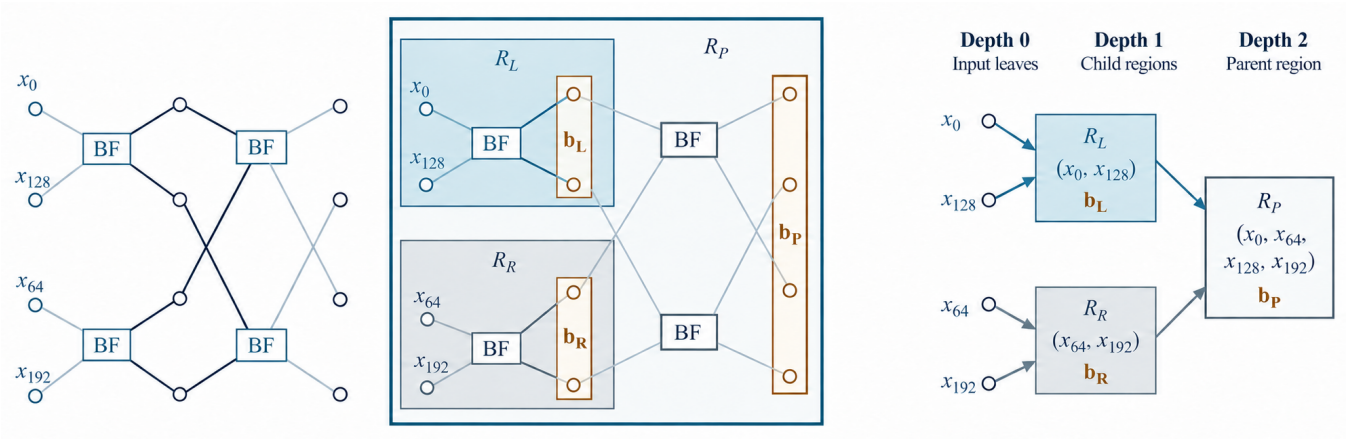
3.1. Hierarchical Inference Tree

Region tree structure. Each node in HIT represents an NTT *region* that spans the computation from a subset of input coefficients through the first n butterfly layers, where the tree depth n ranges from 0 at the leaves to D. Let $R_n^{(j)}$ denote the j-th region at depth n with input index set $\mathcal{I}_n^{(j)}$ of size $|\mathcal{I}_n^{(j)}| = 2^n$ (we omit the superscript j when referring to a generic region). At depth $n = 0$, the leaves correspond to individual input coefficients ($|\mathcal{I}_0| = 1$). At each subsequent level, the butterflies in NTT layer n connect regions at depth n in disjoint pairs, so each parent region R_{n+1} merges two child regions with disjoint inputs $\mathcal{I}_{n+1} = \mathcal{I}_n^{\text{L}} \cup \mathcal{I}_n^{\text{R}}$ (Figure 5).

Because each non-root region has a unique parent at the next depth, this construction yields one binary tree per connected NTT component. At full depth ($n = D$), ML-KEM's seven-layer incomplete NTT ($D = 7$) produces two independent trees of 128 coefficients each (for the even- and odd-indexed inputs, respectively), whereas ML-DSA's eighth butterfly layer ($D = 8$) connects these two components into a single tree over all 256 input coefficients. For any $1 \le n \le D$, we denote the hierarchical representation truncated at depth n by HIT_n.

Candidate assignments and boundary vectors. Let X_i denote the i-th input coefficient with support $\mathcal{X}_i = [-\eta, \eta]$ and assigned value $x_i \in \mathcal{X}_i$. A candidate for region R_n specifies a joint assignment $\mathbf{x}_{\mathcal{I}_n} = (x_i)_{i \in \mathcal{I}_n}$ to all 2^n of its input coefficients, ordered by their original indices. Because the NTT is deterministic, $\mathbf{x}_{\mathcal{I}_n}$ uniquely fixes all intermediate butterfly values within R_n, which we evaluate under the target implementation's exact reduction rules and signed integer representation.

In particular, the 2^n raw integer outputs from R_n form the candidate's *boundary vector* $\mathbf{b}_n$. When two child candidates are combined into a parent candidate $\mathbf{x}_{\mathcal{I}_{n+1}}$, HIT computes the parent boundary $\mathbf{b}_{n+1}$ directly by applying the layer-n butterflies to the two child boundaries $(\mathbf{b}_n^{\text{L}}, \mathbf{b}_n^{\text{R}})$ without reevaluating earlier layers. In Figure 5, for example, candidate assignments to (x_0, x_{128}) and (x_{64}, x_{192}) determine the two child boundaries, which the two layer-1 butterflies map to the parent boundary (see Appendix A for a fully indexed depth-three example).



(a) Scalar factor graph (b) Nested NTT regions (c) Region composition tree

Figure 5. Constructing HIT from a four-input NTT region: (a) scalar factor graph, (b) nested NTT regions, and (c) region composition tree. Circles denote scalar states in (a,b) and input leaves in (c). BF boxes denote butterflies. Dark edges in (a) highlight a cycle, while crossings without circles are not junctions. Blue and gray identify the two child regions. Gold marks the raw output boundary vectors $\mathbf{b}_{\mathrm{L}}$, $\mathbf{b}_{\mathrm{R}}$, and $\mathbf{b}_{\mathrm{P}}$ of the two children and their parent. Each tree node represents a region with joint input candidates. Depths 0, 1, and 2 correspond to input leaves, child regions, and the parent region. Input priors and leakage factors are omitted.

3.2. Likelihood Model and Hierarchical Inference

HIT computes a weight for each candidate from its input prior and the leakage observed within its region. Let $\mathrm{BF}_{t,g}$ denote the g-th butterfly in NTT layer $t \in \{0, \ldots, D-1\}$, and let $\rho \in \mathcal{R} = \{a, b, a', b'\}$ identify a butterfly's input or output role as in Section 2.2. For each butterfly $\mathrm{BF}_{t,g} \in R_n$, a candidate assignment $\mathbf{x}_{\mathcal{I}_n}$ uniquely determines the raw integer value $z_{t,g,\rho}$ at role ρ.

3.2.1. Leakage likelihoods. We write $L_{t,g,\rho}$ for the leakage observed at role ρ of butterfly $\mathrm{BF}_{t,g}$, and $\mathbf{L}$ for all such observations. An intermediate value between adjacent layers serves as an output of one butterfly and an input to another. For example, the output $z_{0,0,a'}$ of $\mathrm{BF}_{0,0}$ is also the input $z_{1,0,a}$ of $\mathrm{BF}_{1,0}$. Because the implementation stores and reloads this shared value across layers, the two roles produce separate leakage observations.

From Equation (2), the local likelihood for a raw intermediate value $z_{t,g,\rho}$ is

$$\ell_{t,g,\rho}(L_{t,g,\rho} \mid z_{t,g,\rho}) = \ell^{\mathrm{HW}}_{t,g,\rho}(L_{t,g,\rho} \mid \mathrm{HW}_w(z_{t,g,\rho})). \quad (6)$$

Because HIT evaluates the exact raw integers fixed by each input candidate, it preserves Hamming weight distinctions among modulo-q congruent values under lazy reduction that likelihoods over $\mathbb{Z}_q$ residues average out.

3.2.2. Bottom-up inference. Because the input coefficients are sampled independently (Section 2.1), the input prior for a candidate assignment is $p_{\mathbf{X}}(\mathbf{x}_{\mathcal{I}_n}) = \prod_{i \in \mathcal{I}_n} p_{X_i}(x_i)$. Under conditional independence of the leakage observations within R_n given $\mathbf{x}_{\mathcal{I}_n}$, the candidate weight is the product of the input prior and the role likelihoods:

$$w_n(\mathbf{x}_{\mathcal{I}_n}; \mathbf{L}) = p_{\mathbf{X}}(\mathbf{x}_{\mathcal{I}_n}) \prod_{\mathrm{BF}_{t,g} \in R_n} \prod_{\rho \in \mathcal{R}} \ell_{t,g,\rho}(L_{t,g,\rho} \mid z_{t,g,\rho}), \quad (7)$$

where the inner product runs over the observed roles $\rho \in \mathcal{R}$ (with $\ell_{t,g,\rho} = 1$ if role ρ is unobserved). These weights define an unnormalized posterior over the region's input assignments. We compute the weights in the log domain and rank candidates by $S_n = \log w_n$.

Inference starts at the leaves with $w_0(x_i; \mathbf{L}) = p_{X_i}(x_i)$ and $\mathbf{b}_0 = x_i$. Leakage from the input roles at $t = 0$ is incorporated when the first butterfly layer is joined. The weight of a parent candidate is the product of the two child weights and the leakage likelihoods from the joining butterfly layer. The joining butterflies map the child boundaries $(\mathbf{b}^{\mathrm{L}}_n, \mathbf{b}^{\mathrm{R}}_n)$ to the raw intermediate values $z_{n,g,\rho}$, which yields the product of likelihoods Λ_{n+1} for the joining layer and the parent weight w_{n+1}:

$$\begin{aligned} \Lambda_{n+1}(\mathbf{b}^{\mathrm{L}}_n, \mathbf{b}^{\mathrm{R}}_n; \mathbf{L}) &= \prod_{\mathrm{BF}_{n,g} \in R_{n+1}} \prod_{\rho \in \mathcal{R}} \ell_{n,g,\rho}(L_{n,g,\rho} \mid z_{n,g,\rho}), \\ w_{n+1}(\mathbf{x}_{\mathcal{I}_{n+1}}; \mathbf{L}) &= w^{\mathrm{L}}_n(\mathbf{x}_{\mathcal{I}^{\mathrm{L}}_n}; \mathbf{L})\, w^{\mathrm{R}}_n(\mathbf{x}_{\mathcal{I}^{\mathrm{R}}_n}; \mathbf{L}) \\ &\quad \times \Lambda_{n+1}(\mathbf{b}^{\mathrm{L}}_n, \mathbf{b}^{\mathrm{R}}_n; \mathbf{L}). \end{aligned} \quad (8)$$

Figure 6 shows how the parent boundary and accumulated score are computed when two child candidates are merged.

Proposition 1 (candidate weights). For any generated candidate, the recursive weight in (8) equals the weight defined in (7). With exhaustive support, normalization gives the posterior for the region's observations.

Proof: At $n = 0$, the region contains no butterflies, so (7) reduces to the initialized leaf weight $p_{X_i}(x_i)$. Assume that the recursive weights of both child regions at depth n agree with (7). Because their input and butterfly sets are disjoint, the product $w^{\mathrm{L}}_n w^{\mathrm{R}}_n$ combines the prior and likelihood factors of both child regions. The factor Λ_{n+1} supplies the likelihoods of the joining layer, so (8) yields the weight defined in (7) for the parent region at depth $n+1$. Normalization over all input assignments gives the posterior. □

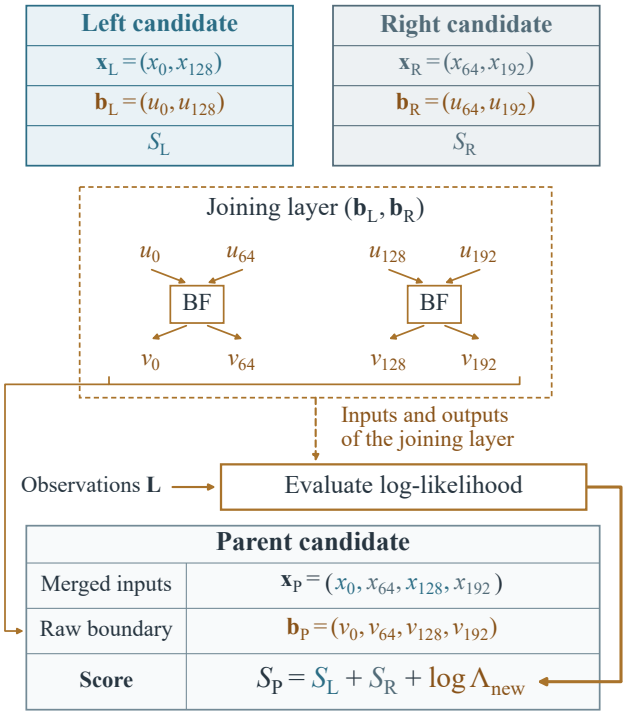


Figure 6. One candidate merge and its accumulated log score $S = \log w$. Each candidate groups an input assignment, a raw boundary, and a score. The parent assignment follows the original input index order, and each joining butterfly (BF) takes one value from each child boundary. The center box encloses the joining layer. Child scores S_L and S_R sum the log-prior and log-likelihood terms of their regions, while the joining step adds the log-likelihood $\log \Lambda_{\text{new}}$ ($\log \Lambda_{n+1}$ in Equation (8)) from the joining layer to form S_P.

Appendix A.1 interprets this recursion in terms of boundary messages and variable elimination.

Root-level recovery. Once bottom-up inference reaches a target depth d ($1 \le d \le D$, with $d = D$ by default), we estimate the input coefficients of each region $R_d^{(j)}$ from its generated candidate set $\mathcal{C}_d^{(j)}$. The coefficient-wise maximum a posteriori (MAP) estimate of X_i is

$$\hat{x}_i = \arg\max_{a \in \mathcal{X}_i} \sum_{\substack{\mathbf{x} \in \mathcal{C}_d^{(j)} \\ x_i = a}} w_d^{(j)}(\mathbf{x}; \mathbf{L}). \tag{9}$$

HIT computes all coefficient estimates directly from the root candidates and completes inference in a single bottom-up pass. An alternative is joint MAP, which selects the single most probable complete input assignment (compared with coefficient-wise MAP in Section 4.4.6).

3.3. Pruning and Computational Cost

At each internal depth $1 \le n < d$, HIT bounds the growth of the search space and retains the K_n highest-scoring candidates per region to form the pruned set $\mathcal{P}_n^{(j)} = \mathrm{Top}_{K_n}(\mathcal{C}_n^{(j)})$. At depth $n+1$, each parent region pairs the candidates from its two child sets $\mathcal{P}_n^{(2j)}$ and $\mathcal{P}_n^{(2j+1)}$ to produce the parent candidate set $\mathcal{C}_{n+1}^{(j)}$, which contains at most K_n^2 candidates. At the target depth d, HIT uses all generated candidates in $\mathcal{C}_d^{(j)}$ to compute the coefficient-wise MAP estimates via Equation (9).

With $2\eta{+}1$ possible values per input coefficient, a depth-n region has $(2\eta{+}1)^{2^n}$ possible input assignments. At shallow depths $n \in \{1, 2\}$, each region spans only one or two butterfly layers and provides limited cross-layer constraints under noise. We therefore retain all candidates at depths $n \in \{1, 2\}$ by setting $K_1 = (2\eta+1)^2$ and $K_2 = (2\eta+1)^4$. Under this setting, depth $n = 3$ evaluates all $K_2^2 = (2\eta{+}1)^8$ eight-input candidates across the first three butterfly layers ($5^8 = 390{,}625$ for $\eta = 2$, $7^8 \approx 5.8 \times 10^6$ for $\eta = 3$, and $9^8 \approx 4.3 \times 10^7$ for $\eta = 4$), whereas an unpruned merge at depth $n = 4$ would require $(2\eta + 1)^{16}$ evaluations ($5^{16} \approx 1.5 \times 10^{11}$ for $\eta = 2$). To keep deeper merges tractable, we apply candidate pruning at all depths $3 \le n < d$ with width K_n (denoted simply by K when the same limit applies across depths $n \ge 3$). Algorithm 3 summarizes the complete inference procedure. Because HIT prunes candidates from their scores within the current region alone, discarded assignments are permanently excluded from subsequent joins. A larger width K_n preserves more candidates for evaluation against later layers at the cost of higher merge overhead.

Algorithm 3 HIT inference to depth d

Require: Likelihoods ℓ, priors p_{X_i}, depth d, widths $\{K_n\}_{n=1}^{d-1}$
Initialize $\mathcal{P}_0^{(i)} = \mathcal{X}_i$, $w_0 = p_{X_i}$, $\mathbf{b}_0 = x_i$
for $n = 1, \ldots, d$ **do**
  for each region $R_n^{(j)}$ **do**
    Join its two child sets to form $\mathcal{C}_n^{(j)}$
    Compute weights by (8)
    if $n < d$ **then**
      $\mathcal{P}_n^{(j)} \leftarrow \mathrm{Top}_{K_n}(\mathcal{C}_n^{(j)})$
      Store boundaries of retained candidates
    else
      Recover $\hat{x}_i$, $i \in \mathcal{I}_n^{(j)}$, by (9)
return $\hat{\mathbf{x}}$

Computational cost. At depth $n+1$ ($0 \le n < d$), HIT processes $256/2^{n+1}$ regions with at most K_n^2 candidate pairs per region. Each pair evaluation computes 2^n butterflies (2^{n+1} intermediate outputs), so the candidate evaluation cost at this depth is $O(256K_n^2)$. Setting $K_0 = 2\eta + 1$ for the leaves, the per-trace cost is

$$C_{\text{HIT}}^{\text{eval}} = O\left(256 \sum_{n=0}^{d-1} K_n^2\right). \tag{10}$$

Appendix A.2 analyzes candidate selection costs and memory requirements. For comparison, the BP baseline enumerates q^2 input pairs per full update of a butterfly factor [12] and requires $O(256TDq^2)$ operations per trace over T forward–backward sweeps across D layers. HIT instead performs a single bottom-up pass whose cost depends only on η and K_n, which decouples the runtime from both the

iteration count T and the modulus q. Section 4.4.7 compares the empirical inference times of HIT and BP.

3.4. Extension to Masked NTTs

In a two-share masked NTT [36], [37], the secret input $\mathbf{a}$ is shared as $\mathbf{a} \equiv \mathbf{b} + \mathbf{c} \pmod{q}$, where $\mathbf{b}$ is a uniformly random mask and both shares range over the full domain $\mathbb{Z}_q$. Because the masks span $\mathbb{Z}_q$, we apply HIT locally at depth two and incorporate these four-input regions as factors in a BP graph (HIT+BP). Each depth-two factor marginalizes over the masks for all $(2\eta+1)^4$ secret assignments and eliminates the short cycles across the first two NTT layers.

Local mask marginalization. For a four-input secret assignment $\mathbf{a} \in [-\eta, \eta]^4$ and mask $\mathbf{b} \in \mathbb{Z}_q^4$, let $\Lambda_{\mathrm{loc}}(\mathbf{a}, \mathbf{b})$ denote the product of the leakage likelihoods at the input, internal, and output states of both shares within the region. The average over the uniformly distributed masks gives the marginal candidate likelihood

$$\mathcal{L}(\mathbf{a}) = q^{-4} \sum_{\mathbf{b} \in \mathbb{Z}_q^4} \Lambda_{\mathrm{loc}}(\mathbf{a}, \mathbf{b}). \tag{11}$$

Because the first butterfly layer is invertible over $\mathbb{Z}_q$, the sum over the four mask coefficients $\mathbf{b}$ can be reparameterized as a sum over the four intermediate values after the first layer. In these intermediate variables, Λ_{loc} factors across the four region butterflies into four pairwise terms. This factorization evaluates $\mathcal{L}(\mathbf{a})$ exactly via two $q \times q$ matrix products and a final sum over q^2 terms, which reduces the time complexity per secret candidate from $O(q^4)$ under direct mask enumeration to $O(q^3)$ with $O(q^2)$ working memory (Appendix B.1).

HIT as a BP factor. Each depth-two HIT region forms a BP factor connected to its secret input vector $\mathbf{a}$ and the output boundary vectors of both shares, whose concatenation we denote by $\mathbf{y}$. Because the four mask coefficients each take q values, a fixed secret candidate $\mathbf{a}$ induces q^4 possible joint output boundaries $\mathbf{y}$. To avoid this $O(q^4)$ branching during repeated BP message updates, we construct a sampled representation of each factor. From the intermediate matrices computed during the $O(q^3)$ contraction (Appendix B.3), we draw S masks with probabilities proportional to $\Lambda_{\mathrm{loc}}(\mathbf{a}, \mathbf{b})$ and compute the resulting boundary samples $\mathbf{y}^{(\mathbf{a},s)}$. We weight the empirical boundary frequency by the candidate likelihood $\mathcal{L}(\mathbf{a})$ to obtain the unbiased factor approximation

$$\widehat{\psi}(\mathbf{a}, \mathbf{y}) = \frac{\mathcal{L}(\mathbf{a})}{S} \sum_{s=1}^{S} \mathbf{1}\{\mathbf{y} = \mathbf{y}^{(\mathbf{a},s)}\}, \tag{12}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function. We connect these

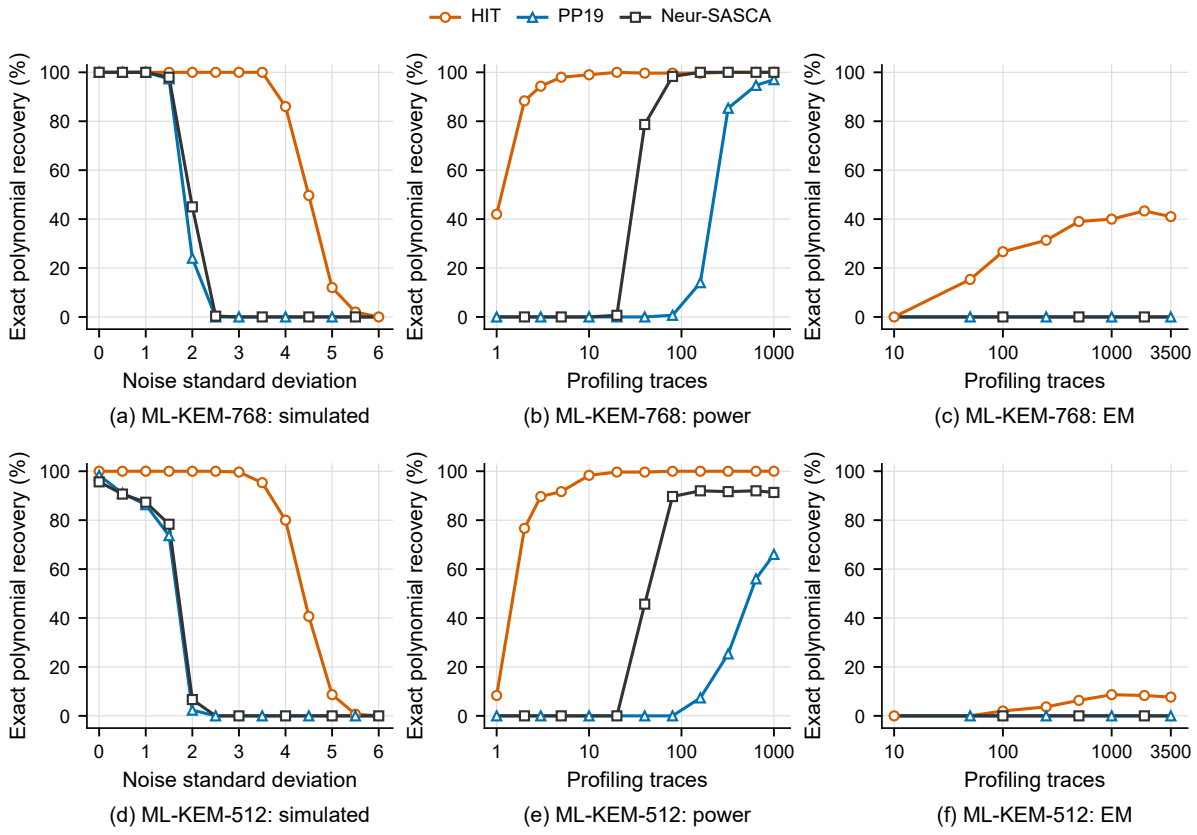


Figure 7. Exact recovery of complete ML-KEM NTT input polynomials. Panels (a)–(c) show ML-KEM-768 and panels (d)–(f) ML-KEM-512 under simulated, power, and EM leakage. The simulated panels share a linear noise axis, while the power and EM panels share logarithmic axes for the number of profiling traces. Each panel compares HIT, PP19, and Neur-SASCA. PP19 and Neur-SASCA achieve 0.0% throughout the EM experiments.

depth-two HIT factors to the butterfly factors in layers $t \geq 2$ through the boundary variables $\mathbf{y}$ and run BP on the resulting graph to obtain coefficient-wise MAP estimates of the secret inputs (Appendix B.2–B.5).

4. Evaluation

We evaluate HIT's recovery performance and computational cost on ML-KEM and ML-DSA to address three research questions:

- **RQ1.** How does HIT compare with existing SASCA methods in single-trace recovery of ML-KEM NTT inputs (Section 4.2)?
- **RQ2.** How effective is HIT for single-trace recovery of ML-DSA NTT inputs (Section 4.3)?
- **RQ3.** How do HIT's design choices affect recovery performance and computational cost (Section 4.4)?

4.1. Experimental Setup

4.1.1. Target implementations and traces. We use the PQClean [26][2] forward NTT implementations for ML-KEM-512, ML-KEM-768, ML-DSA-44, and ML-DSA-65. These parameter sets cover the distinct secret coefficient distributions within each scheme, as ML-KEM-1024 and ML-DSA-87 use the same secret coefficient distributions as ML-KEM-768 and ML-DSA-44, respectively (Table 1).

Simulation. We simulate the NTT and compute the Hamming weights of all intermediate values at butterfly inputs a, b and outputs a', b'. We place the four Hamming weights from each butterfly at fixed positions in a random background signal and apply cubic interpolation to generate a 2,000-sample trace segment. We then add independent Gaussian noise drawn from $\mathcal{N}(0, \sigma^2)$ to each sample.

Physical measurements. We use SAM4S targets on a CW313 carrier board. The target programs are compiled for ARM Cortex-M4 with `-O0` by default, and Section 4.4.4 further evaluates `-O1`, `-O2`, `-O3`, and `-Os` builds. A ChipWhisperer Husky [38] captures power and EM signals at 29.45 MS/s with a 7.37 MHz target clock and uses a Langer LF-B 3 probe for EM measurements. We capture full traces of ML-KEM key generation (Appendix E) and ML-DSA signing, then crop and align the NTT segments so that each extracted segment covers one complete NTT execution. For cross-device recovery, we evaluate all source–target pairs among four devices (A–D), applying templates and PoIs from the source device to attack traces from the target device.

4.1.2. Profiling and comparison methods. In the main experiments, HIT uses Gaussian templates for simulated, power, and EM traces, with a separate mean for each Hamming weight class and a pooled covariance matrix shared across classes [30]. We fit these templates separately for each combination of NTT layer and butterfly role. With 128 butterflies per layer, N profiling NTT traces provide $128N$ observations for a given layer and role. We estimate the shared covariance by averaging the within-class covariance estimates, weighted by their sample counts. All methods use a single PoI for simulated traces and 90 SNR-selected PoIs [27] for power and EM traces. At each simulated noise level, we use the same 400 profiling NTT traces for SNR estimation, PoI selection, and template fitting.

We reproduce PP19 [12] and Neur-SASCA [13] on the same ML-KEM profiling and attack traces as HIT with Wang et al.'s public GPU BP implementation [13],[3] and terminate BP upon message convergence or after 100 iterations. For neural profiling, neural HW posteriors are divided by the class priors in the profiling data to obtain approximate likelihoods. ML-DSA's larger modulus ($q = 8{,}380{,}417$) makes BP inference computationally impractical for PP19 and Neur-SASCA.

4.1.3. Evaluation metrics and inference settings. We evaluate polynomial recovery on 100 attack traces and secret vector recovery on 100 held-out vector groups per run. Success requires all coefficients to be correct, namely 256 coefficients for a polynomial, 768 for ML-KEM-768 $\mathbf{s}$, and 1024 for ML-DSA-44 $\mathbf{s}_1$. We run each condition three times with independently sampled profiling sets and report the mean recovery rate. For polynomial recovery, N denotes the number of profiling NTT traces, whereas for secret vector recovery, N denotes the number of profiling secret vectors (each ML-KEM-768 $\mathbf{s}$ contains three polynomials and each ML-DSA-44 $\mathbf{s}_1$ contains four, so N profiling vectors provide $3N$ or $4N$ NTT traces, respectively). By default, HIT uses all NTT layers and retains $K_n = 1024$ candidates per pruned region.

4.1.4. Masked experiments. We simulate masked ML-KEM-768 with two additive shares and no lazy reduction. Under the leakage model of PP19 [12], we generate one observation per share state from its Hamming weight plus independent $\mathcal{N}(0, \sigma^2)$ noise. HIT+BP uses depth-two regions with $S = 16{,}384$ boundary samples per candidate and terminates BP upon message convergence or after 50 iterations.

4.2. Attacks on ML-KEM

4.2.1. Simulated traces. HIT recovers input polynomials at noise levels where PP19 and Neur-SASCA fail (Figure 7(a,d)). On ML-KEM-768, HIT achieves at least 95.0% polynomial recovery at noise levels up to $\sigma = 3.5$, compared with $\sigma = 1.5$ for Neur-SASCA [13]. At $\sigma = 2.5$, HIT achieves 100.0% mean polynomial recovery for both ML-KEM-512 and ML-KEM-768, while PP19 and Neur-SASCA achieve at most 0.3%. At $\sigma = 4$, HIT still achieves 80.0% recovery for ML-KEM-512 and 86.0% for ML-KEM-768.

2. https://github.com/PQClean/PQClean

3. https://github.com/wyxsjtu/Neur-SASCA

TABLE 2. POWER ATTACKS ON CORTEX-M4 TARGETS USING CHIPWHISPERER BOARDS. THIS WORK REPORTS EXACT RECOVERY OF 256-COEFFICIENT POLYNOMIALS.

Work	Scheme	Target device	Profiling	Attack	Success
MulLeak* [7]	ML-KEM-768	STM32F303	780 traces	12 traces	>99.0%
Qiao et al.* [39]	ML-DSA-44	STM32F405RGTx	200 sigs + 50k traces	1 sig	≈60.0%
Fan et al.* [40]	ML-DSA-44	STM32F415	9,000 traces/component	38 traces	100.0%
PP19† [12]	ML-KEM-512	SAM4S	1,000 NTTs	1 NTT	66.0%
PP19† [12]	ML-KEM-768	SAM4S	1,000 NTTs	1 NTT	97.0%
Neur-SASCA† [13]	ML-KEM-512	SAM4S	1,000 NTTs	1 NTT	91.3%
Neur-SASCA† [13]	ML-KEM-768	SAM4S	160 NTTs	1 NTT	100.0%
HIT	**ML-KEM-512**	**SAM4S**	**80 NTTs**	**1 NTT**	**100.0%**
HIT	**ML-KEM-768**	**SAM4S**	**20 NTTs**	**1 NTT**	**100.0%**
HIT	**ML-DSA-44**	**SAM4S**	**80 NTTs**	**1 NTT**	**100.0%**
HIT	**ML-DSA-65**	**SAM4S**	**80 NTTs**	**1 NTT**	**98.7%**

* Results reported in the cited studies.
† These results are our reproductions of the cited methods.

4.2.2. Power traces. On power traces, HIT achieves higher polynomial recovery rates with fewer profiling traces than PP19 and Neur-SASCA (Figure 7(b,e) and Table 2). For ML-KEM-768, HIT reaches 100.0% mean recovery at $N = 20$, compared with $N = 160$ for Neur-SASCA, while PP19 recovers 97.0% of the polynomials at $N = 1000$ (by comparison, the known-ciphertext profiled attack in MulLeak [7] requires 780 profiling traces and 12 attack traces). For ML-KEM-512, HIT reaches 100.0% recovery at $N = 80$, while Neur-SASCA and PP19 reach 91.3% and 66.0%, respectively, at $N = 1000$. ML-KEM-512 requires more profiling traces than ML-KEM-768 because its wider secret distribution ($\eta_1 = 3$ vs. $\eta_1 = 2$) expands the unpruned depth-three candidate space from $5^8 \approx 3.9 \times 10^5$ to $7^8 \approx 5.8 \times 10^6$.

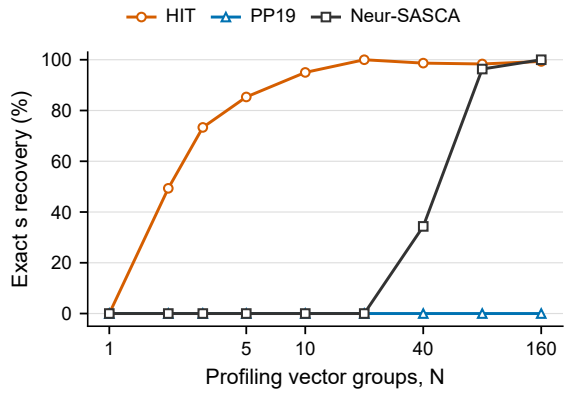


Figure 8. Complete ML-KEM-768 $\mathbf{s}$ recovery from power traces versus profiling vector groups.

4.2.3. Complete secret vector recovery. HIT's profiling advantage extends to recovery of the complete ML-KEM-768 secret vector $\mathbf{s}$ (Figure 8). It achieves 100.0% recovery with 20 profiling vectors, compared with 160 for Neur-SASCA, whereas PP19 recovers no complete vector at 160.

4.2.4. EM traces. Because localized EM measurements exhibit lower SNR than power measurements, both BP-based baselines recover no polynomials across all tested EM budgets (0.0% in Figure 7(c,f)). By contrast, HIT achieves 43.3% recovery for ML-KEM-768 at $N = 2000$ and peaks at 8.7% for ML-KEM-512 at $N = 1000$.

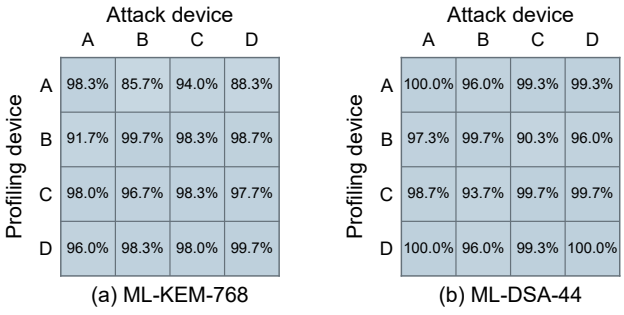


Figure 9. Cross-device exact polynomial recovery with HIT across Devices A–D using power traces. Profiling uses $N = 10$ traces for (a) ML-KEM-768 and $N = 20$ for (b) ML-DSA-44.

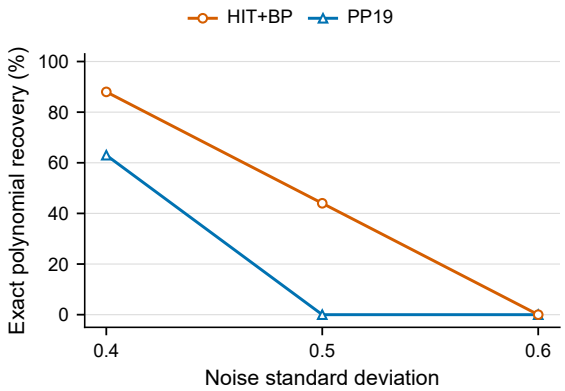


Figure 10. Exact polynomial recovery with HIT+BP and PP19 in two-share masked ML-KEM-768 simulations across noise standard deviations σ.

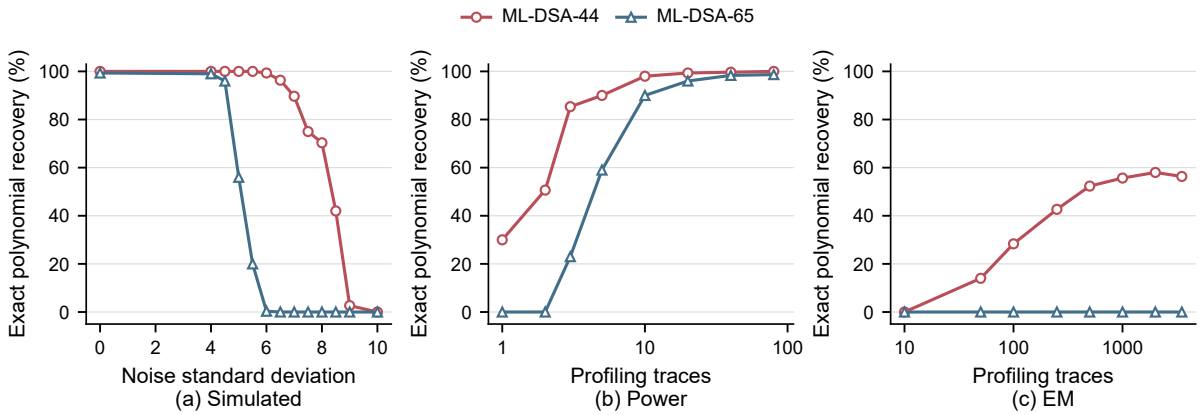


Figure 11. Exact recovery of complete ML-DSA NTT input polynomials with HIT. Panels show (a) simulated traces versus noise level, (b) power traces versus profiling set size, and (c) EM traces versus profiling set size. Each panel compares ML-DSA-44 and ML-DSA-65.

4.2.5. Cross-device evaluation. HIT maintains high polynomial recovery rates when profiling and attack traces come from different devices (Figure 9(a)). With $N = 10$ profiling traces from the source device on ML-KEM-768, HIT achieves 85.7% to 98.7% mean recovery across the 12 cross-device directions among A–D, compared with 98.3% to 99.7% in the same-device setting.

4.2.6. Masked extension. HIT+BP tolerates higher noise levels than PP19 in the two-share masked ML-KEM-768 simulations (Figure 10). At $\sigma = 0.5$, HIT+BP achieves 44.0% recovery of complete input polynomials, compared with 0.0% for PP19.

4.3. Attacks on ML-DSA

4.3.1. Simulated traces. HIT sustains complete polynomial recovery at higher noise levels for ML-DSA-44 than for ML-DSA-65 (Figure 11(a)). ML-DSA-44 maintains 100.0% mean recovery through $\sigma = 5.5$, whereas ML-DSA-65 drops from 99.0% at $\sigma = 4$ to 56.0% at $\sigma = 5$. This gap arises because the wider secret bound of ML-DSA-65 ($\eta = 4$ vs. $\eta = 2$) expands the unpruned depth-three candidate space from $5^8 \approx 3.9 \times 10^5$ to $9^8 \approx 4.3 \times 10^7$.

4.3.2. Power traces. HIT achieves complete or near-complete polynomial recovery for both ML-DSA parameter sets on physical power traces (Figure 11(b) and Table 2). HIT reaches 99.3% mean recovery for ML-DSA-44 at $N = 20$, and with $N = 80$ profiling traces, mean recovery reaches 100.0% for ML-DSA-44 and 98.7% for ML-DSA-65. By comparison, prior physical attacks on ML-DSA-44 either require 200 signatures plus 50,000 traces of profiling to reach $\approx 60.0\%$ single-trace recovery (Qiao et al. [39], Table 2) or require 9,000 profiling traces per polynomial component and 38 attack traces (Fan et al. [40]).

4.3.3. Complete secret vector recovery. HIT's recovery extends to the complete ML-DSA-44 $\mathbf{s}_1$ vector (Figure 12). With 20 profiling vectors, HIT recovers all 100 held-out vectors (100.0%) in every repeat.

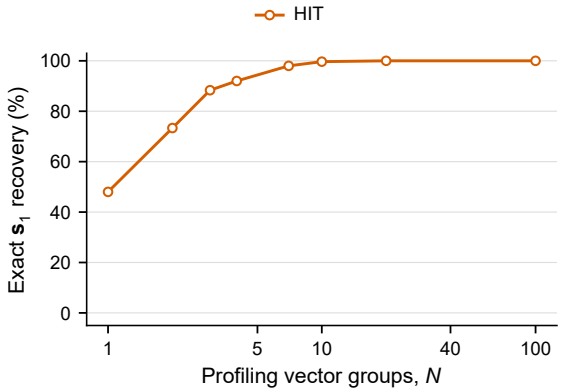


Figure 12. Complete ML-DSA-44 $\mathbf{s}_1$ recovery from power traces versus profiling vector groups, evaluated on 100 held-out groups.

4.3.4. EM traces. On EM traces, HIT achieves 56.3% mean polynomial recovery for ML-DSA-44 at $N = 3500$ (Figure 11(c)). However, recovery remains at 0.0% for ML-DSA-65 across the tested profiling budgets ($N = 10$ to $N = 3500$), as the lower SNR of EM traces is insufficient to retain the true assignment within the top $K = 1024$ candidates out of 9^8 depth-three possibilities.

4.3.5. Cross-device evaluation. Cross-device recovery remains close to same-device recovery for ML-DSA-44 (Figure 9(b)). With $N = 20$ profiling traces from the source device, HIT achieves 90.3% to 100.0% mean recovery across the 12 cross-device directions among A–D, compared with 99.7% to 100.0% in the same-device setting.

4.4. Ablation Studies and Computational Cost

4.4.1. Effect of the profiling method. To isolate the impact of the inference algorithm from that of the leakage model, we pair both TA-based profiling and neural profiling (Neur) with BP and HIT across four combinations (TA+HIT, Neur+HIT, TA+BP, and Neur+BP in Figure 13). Here,

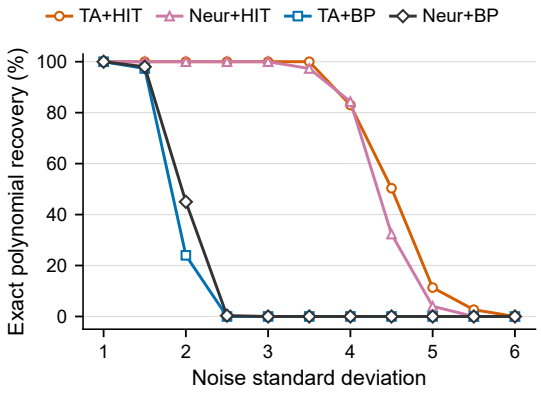


Figure 13. Exact polynomial recovery with TA or neural profiling and BP or HIT inference on simulated ML-KEM-768. TA+BP and Neur+BP correspond to PP19 and Neur-SASCA, respectively.

TA+BP and TA+HIT reuse the fitted templates from our PP19 reproduction, with a separate mean and variance for each HW class. On simulated ML-KEM-768 traces at $\sigma = 2.5$, HIT raises mean polynomial recovery over BP from 0.0% to 100.0% under TA profiling and from 0.3% to 100.0% under neural profiling. At higher noise ($\sigma = 4.5$), the profiling model still influences recovery, where TA+HIT achieves 50.3% compared with 32.3% for Neur+HIT.

4.4.2. Effect of HIT depth. We vary the target inference depth d from one to seven on simulated ML-KEM-768 traces to assess how the number of included NTT layers affects recovery (Figure 14). At shallow depths, HIT lacks sufficient cross-layer leakage constraints to disambiguate all 256 input coefficients. As d increases and HIT combines leakage across more NTT layers, the polynomial recovery rate increases monotonically (see Appendix C for time and memory costs and Appendix D for the depth sweep on physical power traces).

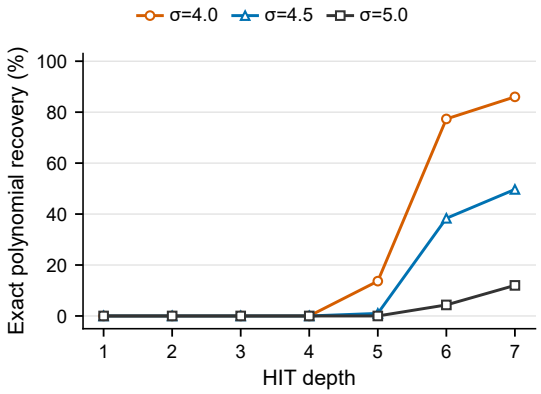


Figure 14. Exact polynomial recovery versus HIT depth for simulated ML-KEM-768. Appendix C reports time and memory; Appendix D gives the depth sweep on measured power traces.

4.4.3. Effect of candidate set size. A larger candidate pruning width K trades additional inference time and memory for higher recovery. On simulated ML-KEM-768 traces, we vary $K_3 = K_4 = K_5 = K_6 = K$ from 64 to 4096 (Figure 15). An increase in K from 1024 to 4096 raises mean recovery from 49.7% to 75.7% at $\sigma = 4.5$ and from 12.0% to 33.7% at $\sigma = 5$. Appendix C reports the corresponding runtime and GPU memory costs.

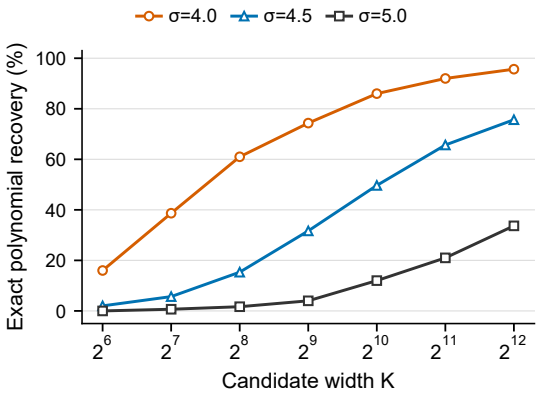


Figure 15. Exact polynomial recovery versus candidate width K for simulated ML-KEM-768.

4.4.4. Effect of compiler optimization. We examine how compiler optimization affects HIT's recovery performance on ML-KEM-768 power traces across five builds (`-O0`, `-O1`, `-O2`, `-O3`, and `-Os` in Figure 16). Because compiler optimizations keep more intermediate values in registers and reduce redundant stack accesses, the optimized builds exhibit lower per-operation leakage SNR and require more profiling traces than `-O0` to fit accurate templates. Nevertheless, HIT achieves 100.0% polynomial recovery with $N = 20$ for `-O0` and with $N = 320$ for `-O1`, `-O2`, and `-O3`. As the profiling set grows, all optimized builds (including `-Os`) approach the recovery performance of `-O0`.

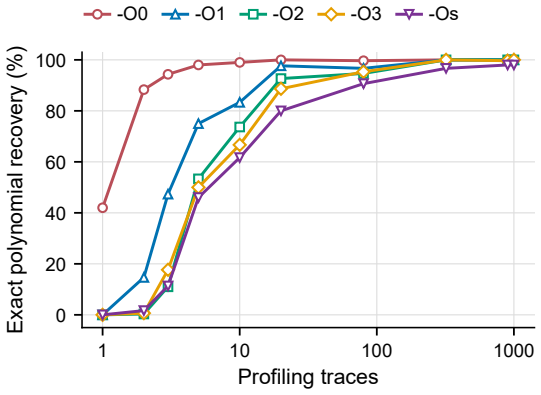


Figure 16. Exact polynomial recovery versus profiling set size for ML-KEM-768 power traces under different compiler optimization levels.

4.4.5. Effect of noise on inference time. Figure 17 shows that HIT's inference time remains nearly constant across noise levels at 0.170–0.172 s for ML-KEM-768 and

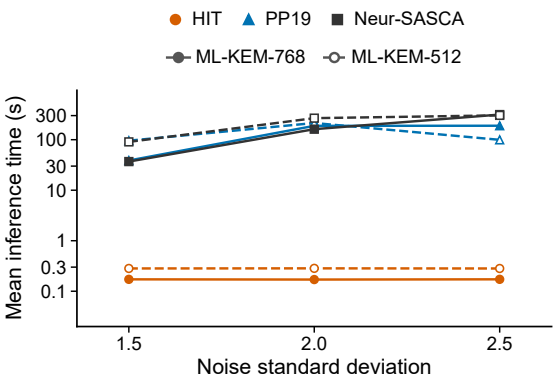


Figure 17. Mean inference time for HIT, PP19, and Neur-SASCA on simulated ML-KEM-768 and ML-KEM-512.

0.282–0.284 s for ML-KEM-512. Across the same noise range ($\sigma \in [1.5, 2.5]$), PP19 requires 38.7–189.0 s and 95.6–214.0 s, respectively, while Neur-SASCA requires 36.9–314.3 s and 90.5–303.2 s. This contrast arises because HIT executes a single bottom-up pass with a fixed workload, whereas both loopy BP methods require more iterations as noise increases and frequently reach the 100-round limit.

4.4.6. Effect of the MAP decision rule. Coefficient-wise MAP matches the polynomial recovery rate of joint MAP over the generated root candidates in our simulated ML-KEM-768 experiments. Applied to the same traces, both rules achieve 86.0%, 49.7%, and 12.0% polynomial recovery at $\sigma = 4, 4.5,$ and 5, respectively.

TABLE 3. INFERENCE RESULTS ON ML-KEM-768 SIMULATED NTT INPUTS AT $\sigma = 2.0$. TIMINGS ARE MEASURED ON RTX 4090 GPUS AND EXCLUDE TEMPLATE FITTING AND ONE-TIME SETUP.

Metric	PP19 BP	Neur-SASCA BP	**HIT**
Exact recovery (/100)	24	45	**100**
Mean BP rounds	71.67	64.42	—
Attacks reaching round 100 (/100)	58.67	48.67	—
Mean time per round/pass (s)	2.6235	2.5072	**0.1703**
Mean inference time (s)	188.036	161.515	**0.1703**
Speedup over PP19	1.00×	1.16×	**1104.37×**

4.4.7. Computational cost. HIT achieves higher recovery at orders of magnitude lower inference cost than both BP-based baselines (Table 3). At $\sigma = 2.0$, HIT recovers all 100 ML-KEM-768 polynomials with a mean inference time of 0.1703 s per polynomial, whereas PP19 (188.036 s) and Neur-SASCA (161.515 s) recover only 24 and 45 polynomials, respectively. Overall, HIT achieves a 1104.37× speedup over PP19 and a 948× speedup over Neur-SASCA, and its full pass is roughly 15× faster than even a single BP iteration (2.51–2.62 s).

Because HIT's complexity is independent of q (Equation (10)), inference remains equally practical for the 23-bit modulus of ML-DSA ($q = 8{,}380{,}417$). Across the tested noise levels, mean inference times are approximately 0.285 s per polynomial for ML-DSA-44 and 0.989 s for ML-DSA-65, where the higher cost for ML-DSA-65 reflects its larger unpruned depth-three candidate space ($\eta = 4$ vs. $\eta = 2$).

5. Discussion

Profiling quality and recovery. In our simulated ablation, TA+HIT outperforms Neur+HIT at $\sigma = 4.5$ (50.3% vs. 32.3%; Section 4.4.1). Standard classification accuracy treats all misclassified samples alike, whereas HIT multiplies likelihoods across many NTT states. A Gaussian template can mispredict the mode yet retain substantial probability on the true Hamming weight. An overconfident neural model [41] can assign near-zero support and penalize the corresponding candidates across joins. Profiling quality therefore requires calibration of full likelihood distributions [41], [42], beyond top-1 Hamming weight accuracy.

Choice of candidate width. At $\sigma = 4.5$, increasing K from 1024 to 4096 raises ML-KEM-768 recovery from 49.7% to 75.7%, with roughly 3.8× runtime and 12.4× memory (Section 4.4.3; Appendix C). Our main evaluations fix $K = 1024$ to balance recovery and cost. Adaptive widths could instead allocate a fixed budget toward ambiguous regions or larger secret domains, such as ML-DSA-65, and prune more aggressively elsewhere.

Implications for countermeasures. Compiler optimizations alone do not prevent recovery (Section 4.4.4), so dedicated countermeasures remain necessary. Fine-grained butterfly shuffling [34], [43] within layers $t \in \{0, 1, 2\}$ obscures the fixed pairings used to construct HIT's initial candidate regions and disrupts deterministic boundary evaluation. Whereas HIT+BP marginalizes two-share arithmetic masks at depth two in $O(q^3)$ time per candidate (Section 3.4), $m > 2$ shares [36], [37] expand the local mask space to $q^{4(m-1)}$ and make exact marginalization prohibitive. Combining masking or shuffling with hiding countermeasures further reduces SNR and impedes early candidate ranking, especially for larger secret bounds η.

6. Conclusion

We introduced HIT for single-trace NTT side-channel attacks. HIT exploits small secret domains and recursive radix-2 butterflies to replace iterative $O(q^2)$ BP with one bottom-up pass and top-K pruning over an acyclic region tree. On ML-KEM-768, HIT runs over 900× faster than PP19 and Neur-SASCA, more than doubles noise tolerance, and achieves 100.0% exact polynomial recovery with 20 Cortex-M4 power profiling traces (8× fewer). Its complexity is independent of q, which enables the first single-trace analytical attack on the ML-DSA NTT ($q = 8{,}380{,}417$). With 80 Cortex-M4 power profiling traces, HIT achieves 100.0% and 98.7% exact polynomial recovery for ML-DSA-44 and ML-DSA-65, respectively. It also fully recovers the secret vectors $\mathbf{s}$ in ML-KEM-768 and $\mathbf{s}_1$ in ML-DSA-44.

References

[1] National Institute of Standards and Technology, "Module-lattice-based key-encapsulation mechanism standard," U.S. Department of Commerce, Federal Information Processing Standards Publication NIST FIPS 203, Aug. 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203

[2] ——, "Module-lattice-based digital signature standard," U.S. Department of Commerce, Federal Information Processing Standards Publication NIST FIPS 204, Aug. 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.204

[3] P. Ravi, S. Sinha Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 307–335, 2020. [Online]. Available: https://doi.org/10.46586/tches.v2020.i3.307-335

[4] V. Q. Ulitzsch, S. Marzougui, M. Tibouchi, and J.-P. Seifert, "Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all," in *Selected Areas in Cryptography (SAC 2022)*, ser. Lecture Notes in Computer Science, vol. 13742. Springer, 2024, pp. 3–32. [Online]. Available: https://doi.org/10.1007/978-3-031-58411-4_1

[5] Z. Chen, E. Karabulut, A. Aysu, Y. Ma, and J. Jing, "An efficient non-profiled side-channel attack on the CRYSTALS-Dilithium post-quantum signature," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 583–590.

[6] T. Tosun and E. Savaş, "Zero-value filtering for accelerating non-profiled side-channel attack on incomplete NTT-based implementations of lattice-based cryptography," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 3353–3365, 2024. [Online]. Available: https://doi.org/10.1109/TIFS.2024.3359890

[7] F. Huang, X. Duan, C. Hu, M. Zheng, and H. Hu, "MulLeak: Exploiting multiply instruction leakage to attack the stack-optimized Kyber implementation on Cortex-M4," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 2, pp. 23–68, 2025. [Online]. Available: https://doi.org/10.46586/tches.v2025.i2.23-68

[8] Z. Xu, O. Pemberton, S. S. Roy, D. F. Oswald, W. Yao, and Z. Zheng, "Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber," *IEEE Trans. Computers*, vol. 71, no. 9, pp. 2163–2176, 2022. [Online]. Available: https://doi.org/10.1109/TC.2021.3122997

[9] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma, "Curse of re-encryption: A generic Power/EM analysis on post-quantum KEMs," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, no. 1, pp. 296–322, 2022. [Online]. Available: https://doi.org/10.46586/tches.v2022.i1.296-322

[10] R. Wang, K. Ngo, J. Gärtner, and E. Dubrova, "Unpacking needs protection: A single-trace secret key recovery attack on Dilithium," *IACR Commun. Cryptol.*, vol. 1, no. 3, p. 12, 2024. [Online]. Available: https://doi.org/10.62056/a0fh89n4e

[11] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 513–533. [Online]. Available: https://doi.org/10.1007/978-3-319-66787-4_25

[12] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *Progress in Cryptology (LATINCRYPT)*, ser. Lecture Notes in Computer Science, P. Schwabe and N. Thériault, Eds., vol. 11774. Springer, 2019, pp. 130–149. [Online]. Available: https://doi.org/10.1007/978-3-030-30530-7_7

[13] Y. Wang, J. Yu, Y. Nie, Y. Zhao, X. Zhang, Y. Shi, C. Zhang, Z. Wang, and D. Gu, "Neur-SASCA: High-noise-tolerant and automated single-trace attack against NTT," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2026, no. 3, pp. 908–932, 2026. [Online]. Available: https://doi.org/10.46586/tches.v2026.i3.908-932

[14] L. Groot Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 21–43, 2018. [Online]. Available: https://doi.org/10.46586/tches.v2018.i3.21-43

[15] P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin, "Side-channel assisted existential forgery attack on Dilithium – a NIST PQC candidate," Cryptology ePrint Archive, Paper 2018/821, 2018. [Online]. Available: https://eprint.iacr.org/2018/821

[16] A. Berzati, A. Calle Viera, M. Chartouny, S. Madec, D. Vergnaud, and D. Vigilant, "Exploiting intermediate value leakage in Dilithium: A template-based approach," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 4, pp. 188–210, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i4.188-210

[17] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, "Soft analytical side-channel attacks," in *Advances in Cryptology (ASIACRYPT)*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8873. Springer, 2014, pp. 282–296. [Online]. Available: https://doi.org/10.1007/978-3-662-45611-8_15

[18] G. Assael, P. Elbaz-Vincent, and G. Reymond, "Improving single-trace attacks on the number-theoretic transform for Cortex-M4," in *Proc. IEEE Int. Symp. Hardware Oriented Security Trust (HOST)*. IEEE, 2023, pp. 111–121. [Online]. Available: https://doi.org/10.1109/HOST55118.2023.10133270

[19] A. Shen, F. Zhang, Y. Jiang, Z. Xin, B. Yang, and W. Xu, "GRALE: Integrating graph-based reduction and auxiliary leakage exploitation for single-trace attack on NTT in ML-KEM," *IEEE Internet Things J.*, vol. 13, no. 19, pp. 45 351–45 368, Oct. 2026. [Online]. Available: https://doi.org/10.1109/JIOT.2026.3717999

[20] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proc. 15th Conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann, 1999, pp. 467–475. [Online]. Available: https://www.cs.ubc.ca/~murphyk/papers/loopy_uai99.pdf

[21] R. Nagpal, G. Cassiers, R. Primas, C. Knoll, F. Pernkopf, and S. Mangard, "On loopy belief propagation for SASCAs: An analysis and empirical study of the inference problem," *IACR Commun. Cryptol.*, vol. 1, no. 4, 2025. [Online]. Available: https://doi.org/10.62056/ayl8ksdja

[22] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 353–367. [Online]. Available: https://doi.org/10.1109/EuroSP.2018.00032

[23] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: A lattice-based digital signature scheme," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 1, pp. 238–268, 2018. [Online]. Available: https://doi.org/10.13154/tches.v2018.i1.238-268

[24] O. Bronchain, M. Azouaoui, M. ElGhamrawy, J. Renes, and T. Schneider, "Exploiting small-norm polynomial multiplication with physical attacks: Application to CRYSTALS-Dilithium," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 2, pp. 359–383, 2024. [Online]. Available: https://doi.org/10.46586/tches.v2024.i2.359-383

[25] Y. Zhou, W. Wang, Y. Sun, and Y. Yu, "Rejected signatures' challenges pose new challenges: Key recovery of CRYSTALS-Dilithium via side-channel attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 4, pp. 817–847, 2025.

[26] M. J. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in *Proc. IEEE Eur. Symp. Security Privacy Workshops (EuroS&PW)*. IEEE, 2022, pp. 19–30. [Online]. Available: https://doi.org/10.1109/EuroSPW55150.2022.00010

[27] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007. [Online]. Available: https://doi.org/10.1007/978-0-387-38162-6

[28] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science, B. S. Kaliski, Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, 2003, pp. 13–28. [Online]. Available: https://doi.org/10.1007/3-540-36400-5_3

[29] S. Bhasin, A. Chattopadhyay, A. Heuser, D. Jap, S. Picek, and R. R. Shrivastwa, "Mind the portability: A warriors guide through realistic profiled side-channel analysis," in *Proc. Network and Distributed System Security Symp. (NDSS)*. The Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/mind-the-portability-a-warriors-guide-through-realistic-profiled-side-channel-analysis/

[30] O. Choudary and M. G. Kuhn, "Efficient template attacks," in *Smart Card Research and Advanced Applications (CARDIS 2013)*, ser. Lecture Notes in Computer Science, vol. 8419. Springer, 2014, pp. 253–270. [Online]. Available: https://doi.org/10.1007/978-3-319-08302-5_17

[31] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, 2001. [Online]. Available: https://doi.org/10.1109/18.910572

[32] M. Hamburg, J. Hermelink, R. Primas, S. Samardjiska, T. Schamberger, S. Streit, E. Strieder, and C. van Vredendaal, "Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 88–113, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i4.88-113

[33] R. C. Rodriguez, F. Bruguier, E. Valea, and P. Benoit, "Cracking the mask: SASCA against local-masked NTT for CRYSTALS-Kyber," *IACR Commun. Cryptol.*, vol. 2, no. 2, 2025. [Online]. Available: https://doi.org/10.62056/aesgbnja5

[34] J. Hermelink, S. Streit, E. Strieder, and K. Thieme, "Adapting belief propagation to counter shuffling of NTTs," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 1, pp. 60–88, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i1.60-88

[35] T. Wedenig, R. Nagpal, G. Cassiers, S. Mangard, and R. Peharz, "Exact soft analytical side-channel attacks using tractable circuits," in *Proc. 41st Int. Conf. Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. A. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 2024, pp. 52 472–52 483. [Online]. Available: https://proceedings.mlr.press/v235/wedenig24a.html

[36] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking Kyber: First- and higher-order implementations," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 173–214, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i4.173-214

[37] M. Azouaoui, O. Bronchain, G. Cassiers, C. Hoffmann, Y. Kuzovkova, J. Renes, T. Schneider, M. Schönauer, F.-X. Standaert, and C. van Vredendaal, "Protecting Dilithium against leakage: Revisited sensitivity analysis and improved implementations," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 4, pp. 58–79, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i4.58-79

[38] C. O'Flynn and Z. D. Chen, "ChipWhisperer: An open-source platform for hardware embedded security research," in *Constructive Side-Channel Analysis and Secure Design (COSADE)*, ser. Lecture Notes in Computer Science, E. Prouff, Ed., vol. 8622. Springer, 2014, pp. 243–260. [Online]. Available: https://doi.org/10.1007/978-3-319-10175-0_17

[39] Z. Qiao, Y. Liu, Y. Zhou, Y. Zhao, and S. Chen, "Single trace is all it takes: Efficient side-channel attack on Dilithium," Cryptology ePrint Archive, Paper 2024/512, 2024. [Online]. Available: https://eprint.iacr.org/2024/512

[40] H. Fan, H. Zhang, Y. Wang, W. Wang, Y. Zhu, H. Zhang, and Q. Yuan, "Screening least square technique assisted multivariate template attack against the random polynomial generation of Dilithium," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 7118–7132, 2024. [Online]. Available: https://doi.org/10.1109/TIFS.2024.3430854

[41] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proc. 34th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1321–1330. [Online]. Available: https://proceedings.mlr.press/v70/guo17a.html

[42] L. Masure, C. Dumas, and E. Prouff, "A comprehensive study of deep learning for side-channel analysis," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 1, pp. 348–375, 2020. [Online]. Available: https://doi.org/10.46586/tches.v2020.i1.348-375

[43] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay, "On configurable SCA countermeasures against single trace attacks for the NTT - A performance evaluation study over Kyber and Dilithium on the ARM Cortex-M4," in *Security, Privacy, and Applied Cryptography Engineering - 10th International Conference, SPACE 2020, Kolkata, India, December 17-21, 2020, Proceedings*, ser. Lecture Notes in Computer Science, L. Batina, S. Picek, and M. Mondal, Eds., vol. 12586. Springer, 2020, pp. 123–146. [Online]. Available: https://doi.org/10.1007/978-3-030-66626-2_7

[44] R. Dechter, "Bucket elimination: A unifying framework for reasoning," *Artif. Intell.*, vol. 113, no. 1-2, pp. 41–85, 1999. [Online]. Available: https://doi.org/10.1016/S0004-3702(99)00059-4

Appendix A. Supplementary Unmasked HIT Derivations

A.1. Boundary Messages and Variable Elimination

For a region R_n, write $\mathbf{x}$ for its input assignment. Deterministic execution fixes every internal variable and the boundary $\beta_n(\mathbf{x})$. Summing the region's factor product over its internal execution variables therefore gives

$$F_n(\mathbf{x}, \mathbf{b}) = w_n(\mathbf{x}; \mathbf{L})\mathbf{1}\{\mathbf{b} = \beta_n(\mathbf{x})\}.$$

The boundary message sums out the input labels:

$$\mu_n(\mathbf{b}) = \sum_{\mathbf{x}} F_n(\mathbf{x}, \mathbf{b}).$$

Since later modeled factors access the region through $\mathbf{b}$, the parent boundary message is obtained by multiplying the two child messages by the factors in the joining layer and summing over the child boundary variables. Processing child interiors before their joining layer gives a region-based elimination order [44].

Exhaustive candidate support represents this exact boundary message, whereas pruning restricts the sum to retained assignments. HIT retains the input labels needed for the coefficient marginals in (9).

A.2. Selection, Readout, and Storage

Selection and readout. Let J_n bound the number of generated candidates per depth-n region: $J_n = (2\eta+1)^{2^n}$ for $n \leq 3$ and $J_n = K_{n-1}^2$ thereafter. Sorting the candidate scores gives the selection bound $O\Big(\sum_{n=3}^{d-1}(256/2^n)J_n \log J_n\Big)$.

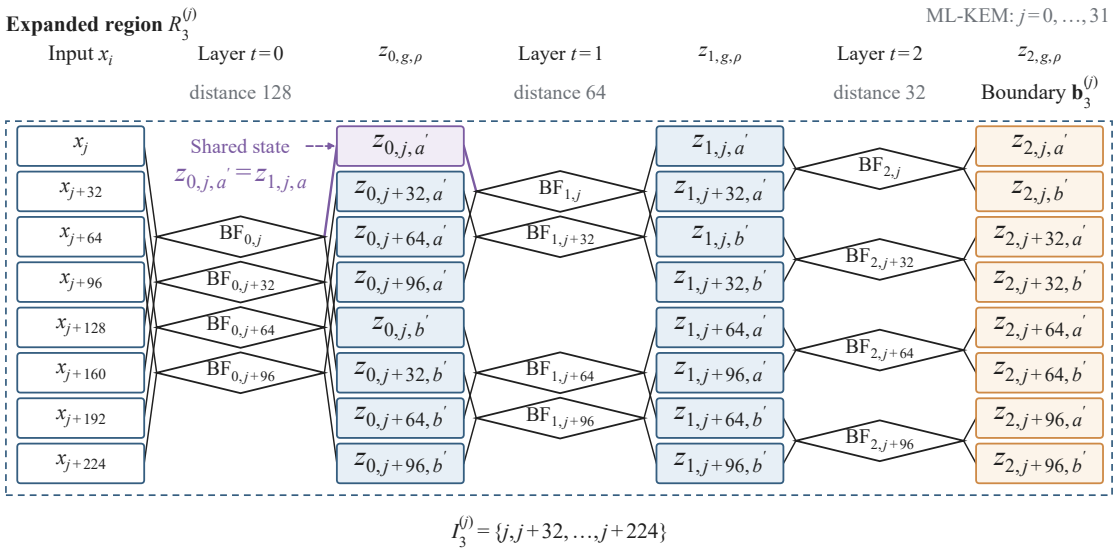


Figure A1. A depth-three HIT region $R_3^{(j)}$ for ML-KEM. For $j = 0, \ldots, 31$, the region contains eight input coefficients x_{j+32k} $(k = 0, \ldots, 7)$ and four butterflies in each of the first three NTT layers.

Root marginal accumulation costs at most $O(256J_d)$, which is covered by the bound on candidate evaluation.

Candidate storage. The implementation stores one scalar score per generated pair and constructs full assignments and output boundaries for retained candidates. A depth-n region uses $O(2^n|\mathcal{P}_n|)$ storage for these states. A join additionally stores J_n scores. Root readout sums candidate weights along the two child dimensions and accumulates the resulting masses by input value.

Appendix B. Masked HIT+BP Extension

B.1. Exact Local Mask Marginalization

For the depth-two region in Section 3.4, fix a secret candidate $\mathbf{a}$ and its local observations. We marginalize over the masks to obtain its likelihood $\mathcal{L}(\mathbf{a})$ from the local likelihood product $\Lambda_{\mathrm{loc}}(\mathbf{a}, \mathbf{b})$ defined there.

Let cut τ be the state after τ butterfly layers. The first-layer map T_1 is bijective over $\mathbb{Z}_q^4$ for odd prime q and nonzero twiddles, so summing over $\mathbf{v} = T_1(\mathbf{b})$ preserves the uniform mask measure. The other share's boundary is $T_1(\mathbf{a}) - \mathbf{v}$. Order $v_0, \ldots, v_3$ by input index. The child regions couple (v_0, v_2) and (v_1, v_3). The joining butterflies couple (v_0, v_1) and (v_2, v_3). Let $F_{\mathrm{L}}, F_{\mathrm{R}}$ contain both shares' likelihoods at cuts 0 and 1, evaluated through the inverse child butterflies, and $G_{\mathrm{L}}, G_{\mathrm{R}}$ the likelihoods at cut 2. Each state contributes once, using its fixed integer representation. All factors below condition on the input candidate and local observations.

$$\mathcal{L}(\mathbf{a}) = q^{-4} \sum_{v_0,v_1,v_2,v_3} F_{\mathrm{L}}(v_0, v_2) F_{\mathrm{R}}(v_1, v_3) \times G_{\mathrm{L}}(v_0, v_1) G_{\mathrm{R}}(v_2, v_3). \tag{13}$$

Each v_i ranges over $\mathbb{Z}_q$. Regrouping the sums gives

$$\begin{aligned} M(v_0, v_3) &= \sum_{v_2} F_{\mathrm{L}}(v_0, v_2) G_{\mathrm{R}}(v_2, v_3), \\ N(v_0, v_1) &= \sum_{v_3} M(v_0, v_3) F_{\mathrm{R}}(v_1, v_3), \\ \mathcal{L}(\mathbf{a}) &= q^{-4} \sum_{v_0,v_1} N(v_0, v_1) G_{\mathrm{L}}(v_0, v_1). \end{aligned} \tag{14}$$

Substituting the first two lines into the third recovers (13). Two $q \times q$ matrix products and a q^2-term sum thus evaluate the marginal in $O(q^3)$ time and $O(q^2)$ working memory per candidate.

B.2. Joint Boundary Factors

Write $Y(\mathbf{a}, \mathbf{b})$ for the joint boundary defined in Section 3.4. Its exact factor is

$$\psi(\mathbf{a}, \mathbf{y}) = q^{-4} \sum_{\mathbf{b} \in \mathbb{Z}_q^4} \Lambda_{\mathrm{loc}}(\mathbf{a}, \mathbf{b}) \times \mathbf{1}\{\mathbf{y} = Y(\mathbf{a}, \mathbf{b})\}. \tag{15}$$

It satisfies $\sum_{\mathbf{y}} \psi(\mathbf{a}, \mathbf{y}) = \mathcal{L}(\mathbf{a})$. For $\mathcal{L}(\mathbf{a}) > 0$, the conditional mask samples in Section 3.4 yield the empirical factor (12).

B.3. Conditional Boundary Sampling

Set $Z = q^4 \mathcal{L}(\mathbf{a}) > 0$ and draw (v_0, v_1), then v_2 and v_3:

$$\begin{aligned} p(v_0, v_1) &= \frac{N(v_0, v_1) G_{\mathrm{L}}(v_0, v_1)}{Z}, \\ p(v_2 \mid v_0, v_1) &= \frac{F_{\mathrm{L}}(v_0, v_2) \sum_{v_3} G_{\mathrm{R}}(v_2, v_3) F_{\mathrm{R}}(v_1, v_3)}{N(v_0, v_1)}, \\ p(v_3 \mid v_0, v_1, v_2) &= \frac{G_{\mathrm{R}}(v_2, v_3) F_{\mathrm{R}}(v_1, v_3)}{\sum_u G_{\mathrm{R}}(v_2, u) F_{\mathrm{R}}(v_1, u)}. \end{aligned} \tag{16}$$

For prefixes with positive probability, these conditionals multiply to $F_{\mathrm{L}}F_{\mathrm{R}}G_{\mathrm{L}}G_{\mathrm{R}}/Z$. Applying T_1^{-1} yields masks distributed as $\Lambda_{\mathrm{loc}}(\mathbf{a},\mathbf{b})/(q^4\mathcal{L}(\mathbf{a}))$. Independent draws give $\mathbb{E}[\widehat{\psi}] = \psi$. Finite samples restrict the represented boundary support.

B.4. BP Interface and Numerical Settings

Boundary samples remain fixed during BP. Input priors enter as unary factors. Factor-to-variable updates multiply each sample weight $\mathcal{L}(\mathbf{a})/S$ by incoming cavity messages on the other edges. For a message to $a_i = u$, sum over samples and candidates with $a_i = u$, using messages from all boundary variables and the other three secret coefficients. HIT includes leakage at cuts $0, 1, 2$ and outer BP at cuts $3, \ldots, D$, with share residues represented in $\{0, \ldots, q-1\}$.

Incoming boundary cavity messages and outgoing messages on the share boundary variables are regularized as $\widetilde{m}(u) = (1-\epsilon)m(u) + \epsilon/q$, $\epsilon = 10^{-6}$. Messages on the secret variables retain their small domains without this mixture. Normalized BP and interface messages use damping $m^{\mathrm{new}} = 0.9m^{\mathrm{fresh}} + 0.1m^{\mathrm{old}}$. We declare convergence when the maximum message change falls below 0.01.

B.5. Computational Cost

The implemented v_2 sampler uses an $S \times q$ by $q \times q$ matrix product, costing $O(Sq^2)$ time. Together with local contraction, this requires $O(q^3 + Sq^2)$ time and $O(q^2 + Sq)$ temporary storage per candidate. For $A = (2\eta+1)^4$ candidates per region and T BP sweeps,

$$C_{\mathrm{HIT+BP}} = O\Big(64A(q^3+Sq^2)+T\big[256(D-2)q^2+64AS\big]\Big). \tag{17}$$

The terms cover local contraction/sampling, pairwise butterfly updates, and scans of AS samples per local factor. Cached samples and messages use $O(64AS+256Dq)$ storage. Sequential working storage is $O(q^2 + Sq + 64AS + 256Dq)$. Batching multiplies temporary storage by the number of concurrent candidates or factors.

Appendix C. Simulated Inference Cost

The HIT measurements in Table 3 and the depth and width sweeps use the Triton implementation with `float64` scores, one polynomial per synchronized call, and three warm-up calls on RTX 4090 GPUs. HIT uses seven layers and $K = 1024$ in Table 3. For the depth and width sweeps, the median synchronized decode time includes CUDA input conversion, inference, readout, metadata checks, and CPU transfer, while setup, template fitting, file I/O, and JIT are excluded. Memory denotes the maximum CUDA allocation observed over 100 attacks at each of the three noise levels. Table 3 reports the mean time at $\sigma = 2.0$, whereas Table 4 reports medians at $\sigma = 4.0, 4.5, 5.0$.

TABLE 4. SIMULATED ML-KEM-768 INFERENCE COST OVER 100 ATTACKS PER CONDITION.

Depth/width	Time (s) at noise σ 4.0	4.5	5.0	Memory (MiB)
HIT depth ($K = 1024$)				
1	0.044	0.043	0.043	0.88
2	0.088	0.089	0.088	1.23
3	0.115	0.115	0.115	8.93
4	0.146	0.143	0.143	19.94
5	0.161	0.160	0.160	22.46
6	0.170	0.168	0.169	22.48
7	0.173	0.172	0.171	22.50
Candidate width (seven layers)				
64	0.151	0.151	0.152	9.23
128	0.152	0.152	0.151	9.40
256	0.154	0.153	0.153	9.73
512	0.158	0.158	0.158	10.40
1024	0.171	0.172	0.173	22.50
2048	0.263	0.263	0.262	75.75
4096	0.658	0.657	0.656	278.25

For each ML-KEM-768 and ML-DSA-44 simulation benchmark, one-time setup takes less than one minute, including template fitting, likelihood generation, GPU initialization, and JIT compilation.

HIT_6 saves little time or memory over HIT_7 while recovering fewer polynomials. In the width sweep, time changes little through $K = 512$, whereas larger widths increase both time and memory. At a fixed width, the three noise levels have similar runtimes.

Appendix D. HIT Depth Sweep on Measured Power Traces

Figure D1 compares HIT depths $d \in \{1, \ldots, 7\}$ on measured power traces, using $K = 1024$ candidates per pruned region.

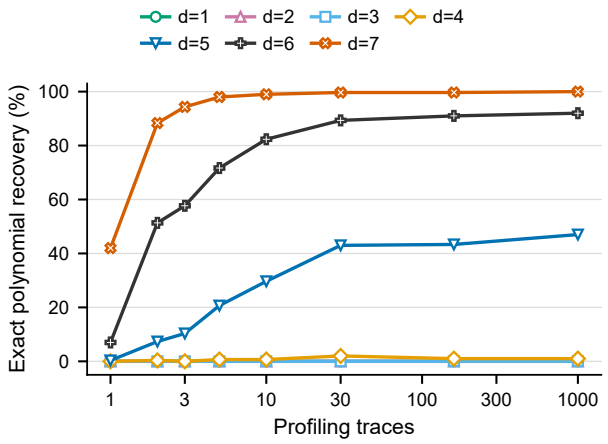


Figure D1. Exact polynomial recovery versus profiling set size for ML-KEM-768 power traces at different HIT depths d.

Appendix E.
Full Power Trace of ML-KEM Key Generation

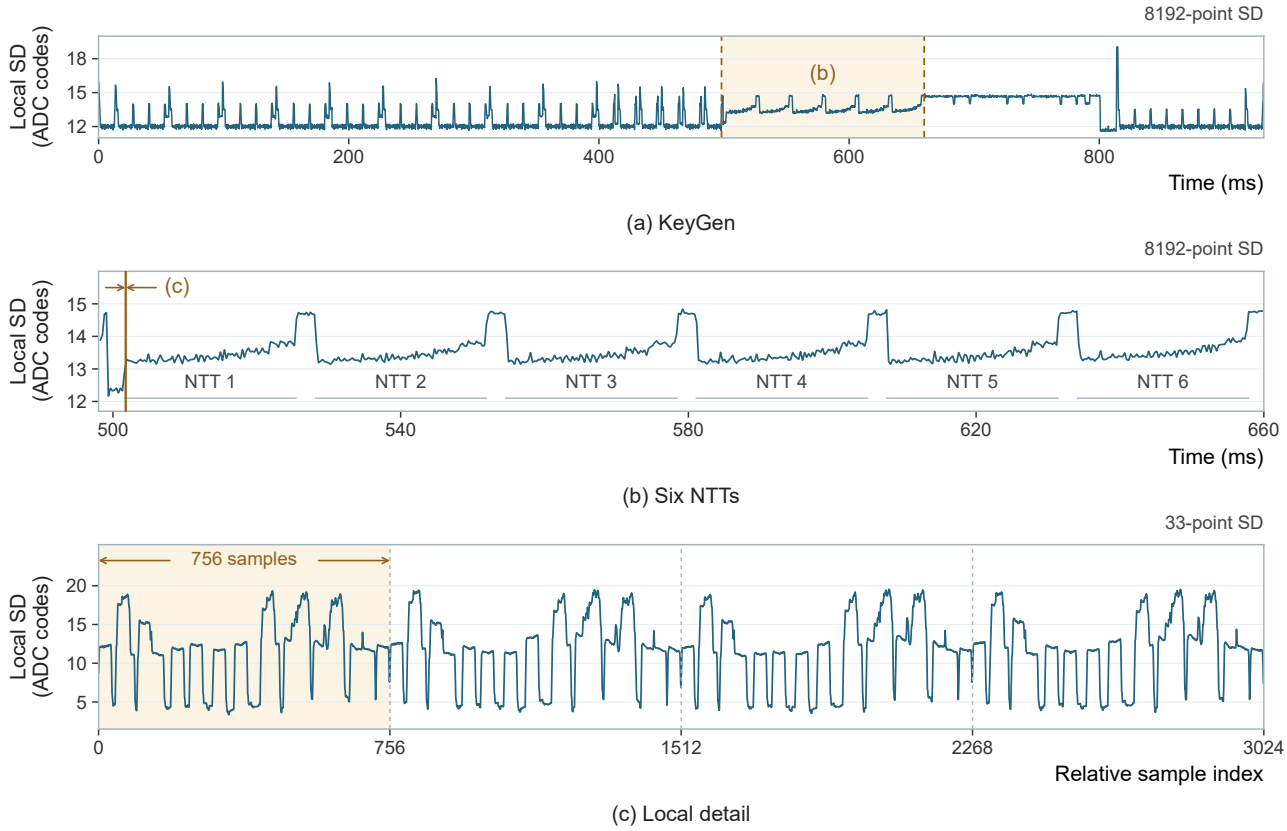


Figure E1. Multiscale views of an ML-KEM-768 KeyGen trace: (a) the full trace, (b) six extracted NTT executions, and (c) a close-up of butterfly operations. Panels (a) and (b) use non-overlapping 8192-sample windows, while (c) uses a centered 33-sample window with unit stride, all reporting population standard deviation in ADC codes.