

Performance and Memory Characterization of NIST Post-Quantum Cryptography Workloads

Saulo M. Z. Freitas, Henrique C. Freitas, Matheus A. Souza

Department of Computer Science
Pontifical Catholic University of Minas Gerais (PUC Minas)
Belo Horizonte, MG, Brazil

saulomzf@gmail.com, {cota,matheusalcantara}@pucminas.br

Abstract. *The advancement of quantum computing threatens widely used public-key cryptography, motivating the post-quantum cryptography (PQC) algorithms selected by NIST for standardization. This work characterizes ML-KEM, ML-DSA, FN-DSA, SLH-DSA, and HQC using the liboqs library, evaluating execution time, key and signature sizes, stack and heap usage, sensitivity to 16 compilation configurations of the benchmark harness, and scalability with message size. ML-KEM is two to three orders of magnitude faster than HQC, ML-DSA offers the most balanced signature performance, and stack usage varies by up to $21\times$ across families while heap peaks differ by at most 47 KiB. For large messages, signing and verification costs converge to the SHAKE-256 throughput, and harness compilation flags show no consistent effect. These results support algorithm selection across deployment scenarios.*

1. Introduction

The performance characterization of cryptographic primitives is a fundamental problem in high-performance computing (HPC). The choice of cryptographic algorithm and compilation strategy directly affects throughput, latency, and memory footprint in security-critical systems.

This problem has become increasingly important with the advancement of quantum computing. By exploiting quantum phenomena such as superposition and entanglement, quantum computers can solve computational problems that are infeasible for classical machines [Preskill 2018; Arute et al. 2019], and the topic has attracted growing attention within the Brazilian high-performance computing community [Rocha et al. 2023; Gabrich et al. 2024; Souza et al. 2025]. As a consequence, algorithms such as RSA and ECC become vulnerable to Shor’s algorithm [Shor 1994], while symmetric cryptography, including AES and SHA, requires larger security parameters to mitigate Grover’s algorithm [Grover 1996].

These advances motivated the U.S. National Institute of Standards and Technology (NIST) to establish the Post-Quantum Cryptography (PQC) standardization process [Alagic et al. 2022]. PQC algorithms run on conventional hardware but are designed to resist attacks by quantum computers. The resulting portfolio includes ML-KEM (CRYSTALS-Kyber), ML-DSA (CRYSTALS-Dilithium), FN-DSA (Falcon), SLH-DSA (SPHINCS+), and, more recently, HQC. These algorithms rely on distinct mathematical foundations, including lattices, hash functions, and error-correcting codes,

leading to substantially different computational and memory behaviors that remain only partially characterized for practical deployment.

Practical deployment depends on multiple factors beyond execution time. These include each algorithm’s response to compiler optimization pipelines, the distribution of memory usage between stack and heap, and the scalability of latency with message size. Existing studies address these dimensions only partially. Most prior work either omits memory analysis, ignores the impact of compilation flags, or evaluates only a subset of the NIST-selected algorithms. As a result, a comprehensive comparative baseline for informed algorithm selection is still lacking.

This gap raises two central research questions:

1. What are the performance differences among the NIST PQC algorithms?
2. What are their implications for real-world deployment in security systems?

This paper addresses these questions through an experimental evaluation of ML-KEM, ML-DSA, FN-DSA, SLH-DSA, and HQC using version 0.10.1 of the *liboqs* library. The evaluation covers five performance dimensions: execution time, cryptographic artifact sizes, memory consumption, the impact of compilation configurations, and signing/verification scalability with message size. All algorithms are evaluated under homogeneous and reproducible conditions on an x86-64 platform with AVX2.

The main contributions of this work are:

- A reproducible experimental methodology for evaluating NIST PQC algorithms across multiple performance dimensions under controlled conditions;¹
- A multidimensional characterization of all five NIST algorithms using representative HPC performance metrics, providing a reproducible baseline for future PQC performance studies;
- An analysis of the relationship between algorithmic design, sensitivity to harness compilation, and architectural characteristics, providing guidelines for selecting PQC algorithms across different scenarios.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details the methodology and implementation. Section 4 presents the results across the five evaluation dimensions. Section 5 discusses limitations and challenges, and Section 6 concludes the paper.

2. Related Work

The transition to PQC represents one of the greatest challenges in digital security. Following the four algorithms selected after the third round, a fourth round selected HQC as a second Key Encapsulation Mechanism (KEM) [Alagic et al. 2025]. Experimental studies evaluating performance under different environments and compilation configurations, however, remain scarce.

Prior work evaluated Kyber, HQC, Dilithium, and Falcon within TLS 1.3 [Sosnowski et al. 2023]; compared Dilithium, Falcon, and SPHINCS+ using *liboqs*, finding Dilithium the most balanced [Opilka et al. 2024]; integrated the

¹Source code and data are available at https://github.com/SauloMFreitas/TCC_PQC

same signatures into PKI and TLS 1.3, reducing handshake time by up to 68% relative to RSA-4096 [Raavi et al. 2025]; compared reference and AVX2 implementations of Kyber and Dilithium on x86-64 [Demir et al. 2025]; and benchmarked Kyber, NTRU, BIKE, Dilithium, and Falcon across server, laptop, and IoT devices [Abbasi et al. 2025].

As Table 1 shows, none of these studies evaluates all five NIST-selected algorithms under homogeneous conditions, only one reports memory, as peak resident set size without separating stack and heap, and none varies compilation flags systematically. This work combines these dimensions with message-size scaling in a single controlled and reproducible environment.

Table 1. Comparison between related work and this work.

Work	Environment	Algorithms	Main Metrics	Compiler Flags	Focus
Sosnowski et al. [2023]	TLS 1.3 (emulated network)	Kyber, HQC, Dilithium, Falcon	Handshake latency, per-library cost	No	PQC in TLS handshakes
Opiłka et al. [2024]	liboqs (Python), VM	Dilithium, Falcon, SPHINCS+	KeyGen, file sign/verify (10 MB–1 GB)	No	Signature performance
Raavi et al. [2025]	liboqs/OpenSSL, VMs	Dilithium, Falcon, SPHINCS+	Sign/verify, message scaling, PKI/TLS	No	PQC signatures in TLS/PKI
Demir et al. [2025]	x86-64	Kyber, Dilithium	KeyGen, Encaps/Decaps, Sign/Verify	No (ref. vs. AVX2)	PQC vs. classical; telecom
Abbasi et al. [2025]	Server, laptop, IoT device	Kyber, NTRU, BIKE, Dilithium, Falcon	Latency, peak memory (RSS), energy, TLS	No	Cross-platform benchmark
This work	x86-64 with AVX2, fixed frequency	ML-KEM, HQC, ML-DSA, FN-DSA, SLH-DSA	Time, artifact sizes, stack/heap, message-size scaling	Yes (harness; 16 configs)	Multi-metric characterization

3. Methodology

3.1. Hardware and Software Environment

Experiments were conducted on a notebook equipped with an AMD Ryzen 5 5500U processor (Zen 2 microarchitecture, 7 nm, 6 physical cores), supporting x86-64, AVX2, FMA, SHA, and AES instructions. The cache hierarchy comprises 192 KiB L1d, 192 KiB L1i, 3 MiB L2, and 8 MiB L3. To favor measurement stability, Simultaneous Multithreading (SMT) and frequency boost were both disabled, holding the processor at its 2.1 GHz base frequency throughout all experiments. The system has 8 GB of DDR4 memory at 3200 MT/s in dual channel, of which about 5.6 GiB is available to

the operating system, the remainder reserved for the integrated graphics. This reservation does not affect the experiments: the largest observed working set (≈ 251 KiB heap peak plus ≈ 177 KiB stack) is four orders of magnitude below the available memory and fits within a single core’s 512 KiB L2 cache. Software: Linux Mint 22.3 (Ubuntu 24.04 base) with kernel 7.0.0-28-generic, GCC 13.3.0, CMake 3.28.3, Ninja 1.11.1, liboqs 0.10.1 (commit 5dd87dc), Valgrind 3.22.0, and Python 3.12.3.

To reduce interference from background processes, all non-essential applications were closed prior to each experiment session. The notebook remained connected to AC power throughout all runs. All timing was performed via `clock_gettime(CLOCK_MONOTONIC)`; no hardware counter interface (`rdtsc` or `perf_event`) was used.

Under these conditions, timing variability is small for operations above $\approx 100 \mu\text{s}$ (HQC, ML-DSA, FN-DSA, SLH-DSA). ML-KEM operations (18–41 μs) lie near the measurement floor (CV below $\approx 14\%$), so absolute ML-KEM times carry modest uncertainty, while relative comparisons remain robust.

3.2. Compilation Flags

The 16 configurations result from combining optimization levels `-O0` through `-O3` with `-march=native` and `-flto`, applied individually and together to the benchmark harness. liboqs was built once as a distributable build (`OQS_DIST_BUILD`, generic optimization target), in which optimized implementations, including AVX2 code paths, are compiled into the library and selected at runtime according to the detected CPU features; SHA-3/SHAKE is provided by liboqs’ own Keccak implementation rather than OpenSSL. The resulting precompiled library was linked identically by all 16 executables, so its cryptographic kernels are the same binary code in every configuration. Since liboqs was not compiled with `-flto`, link-time optimization (LTO) cannot cross the harness–library boundary, and `-march=native` affects only harness code. This experiment therefore measures how sensitive the end-to-end cost of each primitive is to the compilation of the calling code, verifying whether harness compilation choices bias the measurements; recompiling liboqs under each configuration is left as future work.

3.3. Evaluated Algorithms and Variants

All parameterized variants were evaluated: ML-KEM-512/768/1024; ML-DSA-44/65/87; FN-DSA-512/1024; SLH-DSA-SHAKE-128/192/256-s/f; and HQC-128/192/256. The *Open Quantum Safe (OQS) Library (liboqs)* version 0.10.1 was used for all implementations. ML-KEM, ML-DSA, and SLH-DSA were published as FIPS 203, 204, and 205 in August 2024 [National Institute of Standards and Technology 2024a,b,c]; FN-DSA, based on Falcon [Fouque et al. 2020], will be specified in FIPS 206, which is still under development at NIST; HQC was selected in March 2025 and will be specified in FIPS 207 [Alagic et al. 2025]. For brevity, SLH-DSA-SHAKE parameter sets are abbreviated in tables as SLH-128s through SLH-256f.

In liboqs 0.10.1, ML-KEM and ML-DSA correspond to the Initial Public Draft versions of FIPS 203 and FIPS 204, while SLH-DSA and FN-DSA correspond to the Round 3 submissions SPHINCS+ (*simple* variants) and Falcon. The changes introduced in the final standards are minor and do not alter the dominant operations of each scheme, so the performance profiles reported here remain representative.

3.4. Evaluation Procedures

Each operation was executed 200 times after 10 warm-up iterations; for microsecond-range operations, $n = 200$ keeps the relative standard error of the mean ($CV/\sqrt{n}$) below 5% for CV up to 70%. Scalability experiments varied message size from 1 B to 1 MiB in powers of two (21 points) for signature algorithms only, since KEM operations have fixed-size inputs. As the only exception to the iteration count above, the SLH-DSA scalability runs used 100 iterations preceded by 20 warm-up iterations, since signing operations in the hundreds of milliseconds would otherwise make the total runtime of the 21-point sweep intractable; the sub-0.2% Sign CV of SLH-DSA keeps the standard error negligible at this sample size. Memory experiments used a build compiled with `-O0 -g` to preserve allocation behavior and real stack frames; as shown in Section 4.5, the results are identical across all 16 configuration labels. Stack usage was measured via *stack painting* with a 4 MiB sentinel buffer in a dedicated POSIX thread; heap peak via Valgrind `--tool=massif` with one operation per invocation. Throughout this paper, data sizes use binary prefixes (1 KiB = 1024 B, 1 MiB = 1024 KiB), while throughput is reported in decimal MB/s (10^6 B/s), following the usual convention for bandwidth.

3.5. Metrics

Five dimensions were evaluated: (1) **execution time** (mean, standard deviation, median, CV%); (2) **artifact sizes** (public key (PK), secret key (SK), ciphertext (CT), shared secret (SS), and signature (Sig.)); (3) **memory consumption** (stack via stack painting, heap peak via Valgrind massif); (4) **compilation flag impact** (variation relative to `-O0` baseline); and (5) **scalability** (signing/verification time vs. message size). Comparisons are performed at the same NIST security level. The implementation was developed in C using the unified liboqs API (`OQS_KEM` for ML-KEM and HQC; `OQS_SIG` for ML-DSA, FN-DSA, and SLH-DSA), with secret keys zeroed via `OQS_MEM_secure_free()` after each benchmark.

4. Results and Analyses

4.1. Cryptographic Artifact Sizes

Table 2 and Table 3 present artifact sizes. In the KEM group, the HQC-256 ciphertext (14,421 B) is $9.2\times$ larger than ML-KEM-1024 (1,568 B) at level 5, a direct consequence of error-correcting codes requiring larger ciphertexts. In the signature group, FN-DSA produces the most compact signatures (656 B at level 1) due to its compressed NTRU-based encoding. SLH-DSA has the smallest public keys (32–64 B) but the largest signatures (up to 49,856 B), reflecting its hash-only construction. ML-KEM produces a 32-byte shared secret while HQC produces 64 bytes; systems feeding the shared secret into fixed-length key-derivation inputs, such as HKDF-SHA256, must account for this difference.

Table 2. KEM artifact sizes (bytes).

Algorithm	Lvl	PK	SK	CT	SS
ML-KEM-512	1	800	1,632	768	32
ML-KEM-768	3	1,184	2,400	1,088	32
ML-KEM-1024	5	1,568	3,168	1,568	32
HQC-128	1	2,249	2,305	4,433	64
HQC-192	3	4,522	4,586	8,978	64
HQC-256	5	7,245	7,317	14,421	64

Table 3. Digital signature artifact sizes (bytes).

Algorithm	Lvl	PK	SK	Sig.
ML-DSA-44	2	1,312	2,560	2,420
ML-DSA-65	3	1,952	4,032	3,309
ML-DSA-87	5	2,592	4,896	4,627
FN-DSA-512	1	897	1,281	656*
FN-DSA-1024	5	1,793	2,305	1,270*
SLH-128s	1	32	64	7,856
SLH-128f	1	32	64	17,088
SLH-192s	3	48	96	16,224
SLH-192f	3	48	96	35,664
SLH-256s	5	64	128	29,792
SLH-256f	5	64	128	49,856

* Average; variable length.

4.2. Temporal Performance

KEM Algorithms. Table 4 presents KEM execution times. ML-KEM outperforms HQC at all levels: at level 1, ML-KEM-512 KeyGen takes $18.57 \mu\text{s}$ vs. $3,069.37 \mu\text{s}$ for HQC-128 ($\approx 165\times$); at level 5, HQC-256 Decaps reaches $51,979.50 \mu\text{s}$ vs. $39.43 \mu\text{s}$ for ML-KEM-1024 ($\approx 1,318\times$). This advantage stems from lattice arithmetic based on the Number Theoretic Transform (NTT) and native AVX2 assembly in liboqs. HQC cost grows superlinearly with code length (KeyGen $5.6\times$ from HQC-128 to HQC-256, while n grows $3.3\times$). KeyGen, Encaps, and Decaps keep a 1:2:3 ratio at every level, matching their number of polynomial multiplications, which therefore dominate HQC’s cost.

Table 4. KEM performance with $-O0$ (μs , $n = 200$).

Algorithm	Lvl	KeyGen	Encaps	Decaps
ML-KEM-512	1	18.57 ± 1.02	19.49 ± 0.82	18.23 ± 2.51
ML-KEM-768	3	29.74 ± 1.55	29.91 ± 1.73	28.47 ± 2.50
ML-KEM-1024	5	40.62 ± 3.78	40.68 ± 0.54	39.43 ± 1.88
HQC-128	1	$3,069.37 \pm 5.14$	$6,155.21 \pm 12.51$	$9,329.37 \pm 18.70$
HQC-192	3	$9,365.72 \pm 9.88$	$18,777.28 \pm 20.78$	$28,280.25 \pm 19.25$
HQC-256	5	$17,171.70 \pm 62.21$	$34,416.72 \pm 120.15$	$51,979.50 \pm 177.81$

Digital Signature Schemes: ML-DSA and FN-DSA. Table 5 presents lattice-based signature times. FN-DSA covers levels 1 and 5 and ML-DSA levels 2, 3, and 5, so they are directly comparable only at level 5. ML-DSA maintains all three operations in the same order of magnitude. The high Sign CV (61.7% for ML-DSA-44) arises from the *Fiat-Shamir with Aborts* mechanism. The mean exceeds the median by 34.5% for ML-DSA-44, and by 13.2%–16.3% for ML-DSA-65 and ML-DSA-87, confirming that the rejection-sampling bias does not shrink monotonically with the security level. Systems dimensioned by the median can therefore underestimate the average signing cost by up to 34.5%. FN-DSA’s KeyGen is $29\times$ more expensive than Sign at level 1, making it unfavorable for ephemeral keys but competitive when the key pair is reused across multiple signatures.

Digital Signature Schemes: SLH-DSA. Table 5 also presents SLH-DSA times. SLH-DSA records the highest signing cost: up to 1,060.68 ms (s variant, level 3). The

Table 5. Digital signature performance with $-O0$ (μs , $n = 200$).

Algorithm	Lvl	KeyGen	Sign	Verify	CV _{sign}
ML-DSA-44	2	56.41	149.22	55.08	61.7%
ML-DSA-65	3	93.50	227.50	90.86	51.6%
ML-DSA-87	5	154.20	283.58	147.89	47.0%
FN-DSA-512	1	12,045.44	415.37	83.65	1.8%
FN-DSA-1024	5	33,604.67	823.11	168.22	0.9%
SLH-128s	1	81,280	617,160	824.29	0.03%
SLH-128f	1	1,260	29,660	2,155.93	0.10%
SLH-192s	3	118,050	1,060,680	1,169.56	0.05%
SLH-192f	3	1,860	48,010	3,103.31	0.15%
SLH-256s	5	77,410	921,700	1,653.64	0.07%
SLH-256f	5	4,840	97,250	3,117.69	0.12%

f variant reduces this by $9.5\text{--}22.1\times$ depending on the security level, at the cost of larger signatures. Sign CV stays below 0.2% across all variants, confirming the deterministic behavior expected from its construction, in contrast to the stochastic signing of ML-DSA.

Comparison with prior work. These results agree with prior findings that lattice-based signatures incur lower computational cost than SPHINCS+ [Raavi et al. 2025], with ML-DSA offering the best balance [Opilka et al. 2024]. For KEMs, however, HQC was found on par with Kyber at the TLS handshake level [Sosnowski et al. 2023], whereas our isolated measurements reveal a gap of two to three orders of magnitude, possibly amplified by the HQC implementation in liboqs 0.10.1: protocol latency masks primitive costs that matter in compute-bound or constrained deployments.

4.3. Impact of Compilation Configurations

KEM Algorithms. Figure 1 shows that $-O0$ is among the fastest configurations for ML-KEM, since liboqs runs ML-KEM through its own AVX2 assembly, independent of harness flags. Variation stays below 4% for ML-KEM and 1.3% for HQC, whose millisecond-range cost dwarfs any harness overhead.

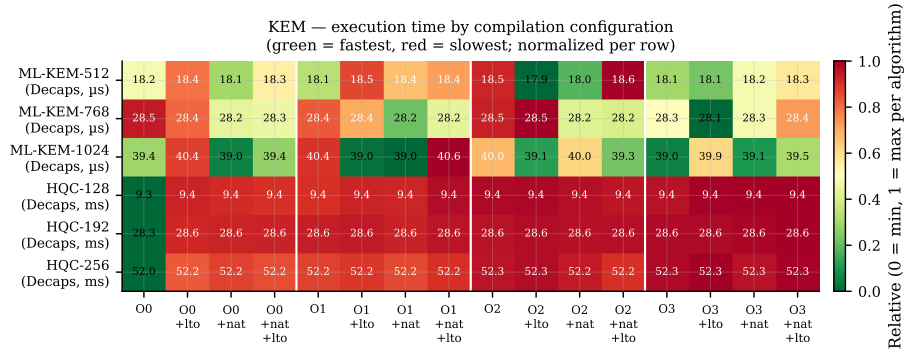


Figure 1. Heat map of KEM Decaps time per compilation configuration. Green = fastest; red = slowest. Values in μs (ML-KEM) and ms (HQC), normalized per row.

ML-DSA and FN-DSA. Figure 2 shows contrasting patterns. ML-DSA’s heat map is irregular because stochastic *Fiat-Shamir with Aborts* variability ($\text{CV} > 45\%$) dominates any flag effect, producing swings of $15\text{--}26\%$ in Sign time that do not track any single flag

consistently. FN-DSA KeyGen varies by at most $\approx 7\%$ with no consistent pattern across parameter sets: for FN-DSA-512 the fastest configuration is plain $-O2$ ($1.024\times$) and the slowest $-O2+nat$ ($0.943\times$), whereas for FN-DSA-1024 the $-O0$ baseline itself is fastest. Since FN-DSA’s computation resides entirely in the precompiled liboqs, these differences cannot result from recompiling its kernels; they are attributed to run-to-run effects such as executable code and data layout, and should not be read as flag-driven speedups.

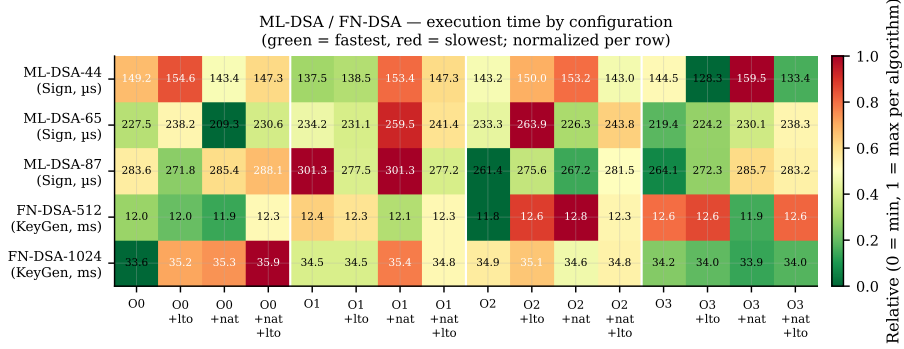


Figure 2. Heat map of Sign time (ML-DSA) and KeyGen time (FN-DSA) per compilation configuration. Irregular ML-DSA pattern and FN-DSA variation without a consistent pattern.

SLH-DSA. Figure 3 shows the flattest pattern among all evaluated families: SLH-DSA-SHAKE signing time varies by less than 0.3% across all 16 compilation configurations, for every variant. This behavior is consistent with the same mechanism identified for ML-KEM above: liboqs computes SHAKE-256, the operation that dominates SLH-DSA’s cost, with its own Keccak implementation, whose optimized variant is selected at runtime and is independent of the optimization flags applied to the surrounding harness code (the Zen 2 microarchitecture has no dedicated SHA-3 instructions). Since the harness contributes only a negligible fraction of the total signing time, no flag combination produces a measurable effect on SLH-DSA.

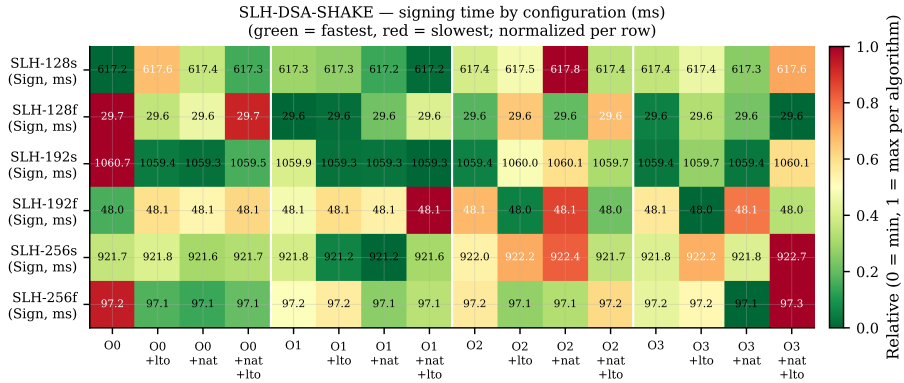


Figure 3. Heat map of SLH-DSA-SHAKE Sign time per compilation configuration. All 16 configurations fall within 0.3% of one another for every variant. Values in ms, normalized per row.

4.4. Scalability with Message Size

All ML-DSA and FN-DSA variants show a constant phase up to ≈ 1 KiB, followed by linear growth at $\approx 6.0 \mu\text{s}/\text{KiB}$ (Figure 4), which also holds for verification (Figure 5). For large messages, cost is therefore governed by a primitive shared by all schemes, absorbing the message into SHAKE-256 at $\approx 170 \text{ MB/s}$ (≈ 12.4 cycles/byte at 2.1 GHz), rather than by each scheme's own mathematical structure.

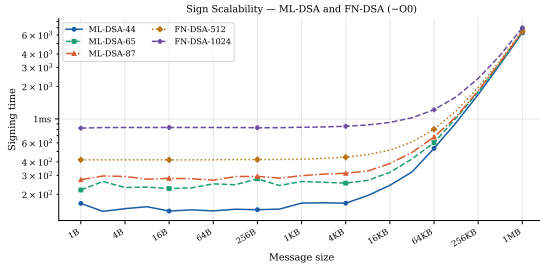


Figure 4. Signing time vs. message size for ML-DSA and FN-DSA (–00). Linear growth $\approx 6.0 \mu\text{s}/\text{KiB}$ from 1 KiB onward, dominated by SHAKE-256.

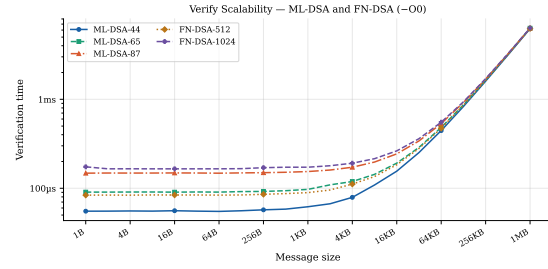


Figure 5. Verification time vs. message size for ML-DSA and FN-DSA. Slope $\approx 6.0 \mu\text{s}/\text{KiB}$ confirms SHAKE-256 as bottleneck.

SLH-DSA's signing time is practically insensitive to message size for the *s* variants (Figure 6), which grow by at most 2.1% from 1 B to 1 MiB. The *f* variants (Figure 7) grow more visibly, from 12.6% (SLH-DSA-SHAKE-256f, 97.3 to 109.5 ms) to 41.5% (SLH-DSA-SHAKE-128f, 29.7 to 42.0 ms), since their proportionally smaller fixed tree-construction cost lets the message-dependent SHAKE-256 term contribute a larger relative share. In both cases the dominant cost remains the Merkle tree construction, which is fixed and message-independent; verification, in contrast, grows linearly for all SLH-DSA variants at $\approx 6.0 \mu\text{s}/\text{KiB}$ (Figure 8).

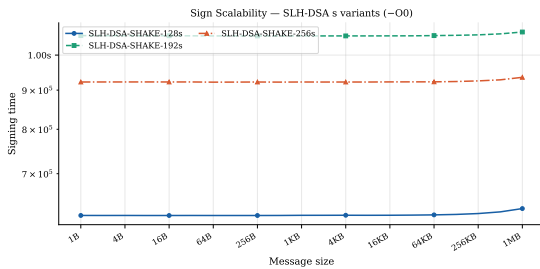


Figure 6. Signing time vs. message size for SLH-DSA *s* variants. Growth $\leq 2.1\%$ for all variants.

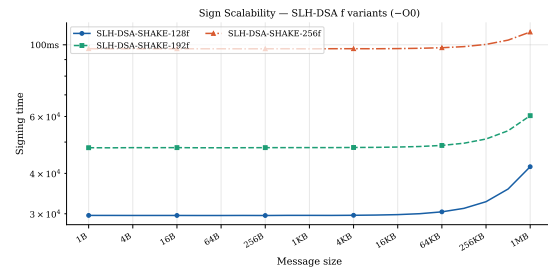


Figure 7. Signing time vs. message size for SLH-DSA *f* variants. Growth from 1 B to 1 MiB ranges from 12.6% (level 5) to 41.5% (level 1).

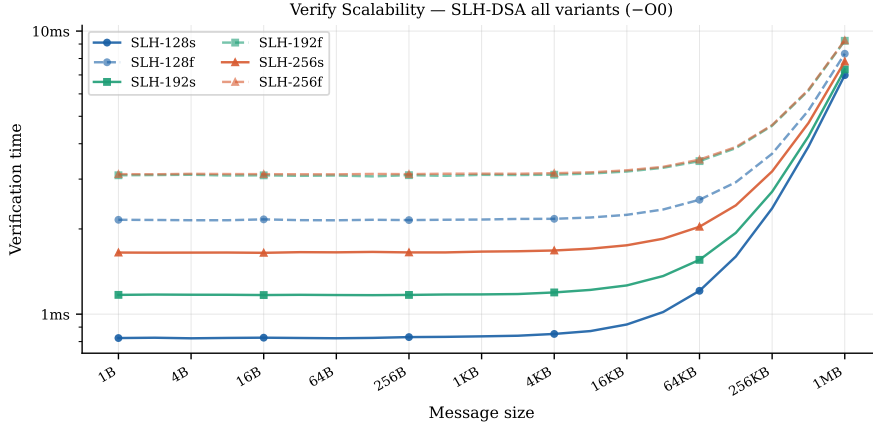


Figure 8. Verification time vs. message size for all SLH-DSA variants. Linear growth $\approx 6.0 \mu\text{s/KiB}$. Solid = *s*; dashed = *f*.

4.5. Memory Consumption

Table 6 presents stack usage and heap peak. HQC-256 requires the largest stack footprint, reaching 181,072 bytes during Decaps ($6.4\times$ more than ML-KEM-1024) due to high-dimensional polynomial operations over its quasi-cyclic code structure during decapsulation. ML-DSA also exhibits high stack usage during Sign, whereas FN-DSA shows a pronounced asymmetry between signing and verification. In contrast, SLH-DSA has the lowest stack consumption among the signature schemes (<30 KiB at level 5), comparable to ML-KEM, despite its substantially higher execution time.

Table 6. Stack usage per operation (bytes) and heap peak (KiB, maximum over the three operations).

Algorithm	Stack (bytes)			Heap peak (KiB)
	KeyGen	Enc/Sign	Dec/Verify	
ML-KEM-512	13,520	16,128	16,928	204.99
ML-KEM-768	17,680	20,816	21,936	207.20
ML-KEM-1024	22,800	26,480	28,080	208.20
HQC-128	38,608	56,480	60,928	211.10
HQC-192	70,720	106,784	115,792	219.76
HQC-256	108,800	166,624	181,072	230.62
ML-DSA-44	28,336	53,488	26,000	208.93
ML-DSA-65	33,488	83,344	29,104	211.86
ML-DSA-87	41,680	117,104	35,504	214.61
FN-DSA-512	23,640	49,616	9,096	204.83
FN-DSA-1024	40,536	90,768	13,192	207.40
SLH-128s	14,512	13,776	8,496	210.33
SLH-128f	14,032	13,392	8,496	219.34
SLH-192s	21,008	19,536	9,360	218.55
SLH-192f	20,272	18,960	9,392	237.53
SLH-256s	29,776	27,344	10,512	231.84
SLH-256f	29,136	26,832	10,544	251.44

Table 6 reveals a clear dichotomy between stack and heap usage. Heap peak ranges from 204.8 to 251.4 KiB. Most of this value is a baseline common to every run (process runtime, liboqs initialization, and the benchmark harness), which was not isolated in this study; the algorithm-dependent component is therefore bounded by the spread between algorithms, at most ≈ 47 KiB. Within SLH-DSA, the *f* variants

consistently require 9–20 KiB more heap than the corresponding *s* variants. In contrast, stack usage differs by up to $21\times$ between algorithm families, making it the primary memory constraint in embedded deployments. HQC presents the highest risk of stack overflow in systems with less than 64 KiB of RAM, whereas SLH-DSA achieves the lowest stack consumption among signature schemes despite its higher computational cost.

Validity of Memory Measurements Across Configurations. Across 816 stack measurements ($17 \text{ algorithms} \times 3 \text{ operations} \times 16 \text{ configuration labels}$), stack usage was byte-for-byte identical to the `-O0` baseline and heap peak deviated by at most 0.6 KiB, within allocator noise; memory results are therefore independent of the compilation configuration.

4.6. Recommendations by Use Case

Based on the experimental results, practical recommendations for x86-64 platforms with AVX2 are as follows. For general-purpose key encapsulation, **ML-KEM** offers the best balance of performance, artifact size, and memory; **HQC** is recommended as the only code-based alternative for mathematical diversification. For digital signatures, **ML-DSA** provides the best operational balance across KeyGen, Sign, and Verify; when dimensioning systems, note that the median reflects typical signing latency, while sustained throughput must be provisioned by the mean, which exceeds the median by up to 34.5%. **FN-DSA** becomes competitive in long-lived key scenarios where the KeyGen cost ($\sim 12.0 \text{ ms}$) is amortized over many signatures, and also favors bandwidth-constrained environments with its 656 B signatures vs. up to 49,856 B for SLH-DSA. Within **SLH-DSA**, the *f* variants offer the lowest absolute signing latency (29.7–97.3 ms vs. 617–1,061 ms for *s*), while the *s* variants offer smaller signatures and signing time nearly independent of message size ($\leq 2.1\%$ growth up to 1 MiB, vs. 12.6–41.5% for *f*). Both variants have the lowest stack footprint among the evaluated signature schemes ($< 30 \text{ KiB}$ at level 5), making SLH-DSA the safest signature choice for devices with severe stack constraints, where HQC should be avoided (177 KiB at Decaps).²

5. Limitations and Challenges

Limitations. The scope of this study is bounded by seven factors. (i) All measurements were taken on a single client-class processor at a fixed 2.1 GHz with boost disabled; absolute times are therefore conservative and may differ on server processors, with larger caches and, in recent generations, AVX-512, or on embedded platforms. (ii) liboqs 0.10.1 implements pre-standard versions (ML-KEM and ML-DSA as Initial Public Drafts, SPHINCS+ and Falcon from Round 3, and HQC prior to FIPS 207). (iii) liboqs was built once, so the 16 configurations affected only the harness, and LTO could not cross the library boundary. (iv) Only the SHAKE-based SLH-DSA parameter sets were evaluated; the SHA-2 variants could benefit from the processor’s SHA extensions. (v) Timing relied on `clock_gettime` without hardware counters, so microarchitectural effects such as cache misses and instructions per cycle were not measured directly. (vi) The heap baseline common to all runs was not isolated. (vii) Primitives were measured in isolation and single-threaded, without protocol context, concurrency, or side-channel validation.

²**Normative status.** FN-DSA (FIPS 206) is still under development and HQC (to be specified as FIPS 207) has not yet been standardized; neither is production-ready. ML-KEM (FIPS 203), ML-DSA (FIPS 204), and SLH-DSA (FIPS 205) are final standards.

Challenges. Four practical challenges shaped the methodology. ML-KEM operations (18–41 μ s) lie near the resolution of `clock_gettime`, requiring warm-up iterations and robust statistics. ML-DSA’s rejection sampling produced Sign CVs above 45%, making it necessary to report the median alongside the mean. The 21-point SLH-DSA message-size sweep, with signing operations of up to ≈ 1 s, required reducing iterations to keep the runtime tractable. Finally, attributing compilation effects required separating harness code from the precompiled library.

6. Conclusion

This work presented a comparative experimental evaluation of ML-KEM, HQC, ML-DSA, FN-DSA, and SLH-DSA under homogeneous and reproducible conditions, providing a multidimensional performance baseline for NIST-selected post-quantum cryptographic algorithms.

The results show that performance differences among PQC algorithms are substantial and depend on the metric and scenario, so no single scheme dominates: ML-KEM is the most efficient KEM and HQC the code-based alternative for diversification, while among signatures ML-DSA offers the best overall balance, FN-DSA suits bandwidth-constrained and long-lived key scenarios, and SLH-DSA has the lowest stack footprint among signature schemes. Memory analysis further revealed that, while heap peaks differ by at most ≈ 47 KiB across all algorithms, stack usage varies by up to $21\times$, making it a critical criterion for resource-constrained deployments.

Compilation experiments showed that harness compilation choices do not materially affect the measured cost of any family: ML-KEM, HQC, and SLH-DSA varied by less than 4% across all 16 configurations, ML-DSA’s variation is dominated by the stochastic behavior of *Fiat-Shamir with Aborts*, and FN-DSA’s differences of up to $\approx 7\%$ follow no consistent pattern. The analysis also identified two relevant findings: a significant, non-monotonic bias between mean and median signing time in ML-DSA, and a common SHAKE-256 throughput limit of ≈ 170 MB/s (≈ 6.0 μ s/KiB) governing the message-dependent cost of signing and verification in all evaluated signature schemes.

Overall, the results answer the research questions posed in this work by showing that PQC algorithms exhibit distinct computational characteristics across representative HPC metrics, leading to different trade-offs for practical deployment. Addressing the limitations discussed in Section 5, future work includes evaluating server and embedded platforms (ARM Cortex-M, RISC-V), recompiling `liboqs` under each configuration while validating constant-time properties with *dudect* [Reparaz et al. 2017] and *ct-verif* [Almeida et al. 2016], including the SHA-2 variants of SLH-DSA, integrating PQC into TLS 1.3 and SSH, and benchmarking hybrid deployments.

Acknowledgments

This research was funded by the Minas Gerais State Research Foundation (FAPEMIG) [APQ-01863-26 and PCE-00344-26] and National Council for Scientific and Technological Development (CNPq) [309494/2026-5] and PUC Minas [FIP-2026/35258]. The authors thank for their support.

Artificial Intelligence Usage Statement

The AI assistant Claude (Anthropic, version Claude Sonnet 4.6 and Opus 5.5) was used for understanding the technologies employed (liboqs, Valgrind, CMakeLists), critical review, inconsistency identification, and writing improvements.

References

- M. Abbasi, F. Cardoso, P. Váz, J. Silva, and P. Martins. A practical performance benchmark of post-quantum cryptography across heterogeneous computing environments. *Cryptography*, 9(2), 2025. ISSN 2410-387X. doi: 10.3390/cryptography9020032. URL <https://www.mdpi.com/2410-387X/9/2/32>.
- G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and Y.-K. Liu. Status report on the third round of the NIST post-quantum cryptography standardization process. NIST Interagency Report 8413-upd1, National Institute of Standards and Technology, 2022. URL <https://doi.org/10.6028/NIST.IR.8413-upd1>.
- G. Alagic, M. Bros, P. Ciadoux, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. A. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, H. Silberg, D. Smith-Tone, and N. Waller. Status report on the fourth round of the NIST post-quantum cryptography standardization process. Technical Report NIST IR 8545, National Institute of Standards and Technology, 2025. URL <https://doi.org/10.6028/NIST.IR.8545>.
- J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, and M. Emmi. Verifying Constant-Time implementations. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 53–70, Austin, TX, Aug. 2016. USENIX Association. ISBN 978-1-931971-32-4. URL <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida>.
- F. Arute, K. Arya, R. Babbush, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. doi: 10.1038/s41586-019-1666-5.
- E. D. Demir, B. Bilgin, and M. C. Onbasli. Performance analysis and industry deployment of post-quantum cryptography algorithms, 2025. URL <https://arxiv.org/abs/2503.12952>.
- P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-fourier lattice-based compact signatures over NTRU, 2020. URL <https://falcon-sign.info/>. Specification v1.2, submission to the NIST Post-Quantum Cryptography Standardization Process.
- M. Gabrich, H. Freitas, and M. Souza. Análise de redes neurais para crispr: Uma abordagem com computação quântica. In *Anais do XXV Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 13–24, Porto Alegre, RS, Brasil, 2024. SBC. doi: 10.5753/sscad.2024.244778. URL <https://sol.sbc.org.br/index.php/sscad/article/view/30982>.
- L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL <https://doi.org/10.1145/237814.237866>.

- National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Federal Information Processing Standards Publication FIPS 203, U.S. Department of Commerce, 2024a. URL <https://doi.org/10.6028/NIST.FIPS.203>.
- National Institute of Standards and Technology. Module-lattice-based digital signature standard. Federal Information Processing Standards Publication FIPS 204, U.S. Department of Commerce, 2024b. URL <https://doi.org/10.6028/NIST.FIPS.204>.
- National Institute of Standards and Technology. Stateless hash-based digital signature standard. Federal Information Processing Standards Publication FIPS 205, U.S. Department of Commerce, 2024c. URL <https://doi.org/10.6028/NIST.FIPS.205>.
- F. Opiřka et al. Performance analysis of post-quantum cryptography algorithms for digital signature. *Applied Sciences*, 14(12), 2024. ISSN 2076-3417. doi: 10.3390/app14124994. URL <https://www.mdpi.com/2076-3417/14/12/4994>.
- J. Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018. doi: 10.22331/q-2018-08-06-79.
- M. Raavi et al. Security and performance analyses of post-quantum digital signature algorithms and their tls and pki integrations. *Cryptography*, 9(2), 2025. ISSN 2410-387X. doi: 10.3390/cryptography9020038. URL <https://www.mdpi.com/2410-387X/9/2/38>.
- O. Reparaz, J. Balasch, and I. Verbauwhede. Dude, is my code constant time? In *Proceedings of the Conference on Design, Automation & Test in Europe, DATE '17*, pages 1701–1706, Leuven, BEL, 2017. European Design and Automation Association.
- C. Rocha, A. Santos, and M. Souza. A systematic literature review on optimization techniques for quantum computing compilers. In *Anais Estendidos do XXIV Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 25–32, Porto Alegre, RS, Brasil, 2023. SBC. doi: 10.5753/wscad_estendido.2023.235804. URL https://sol.sbc.org.br/index.php/sscad_estendido/article/view/26445.
- P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Symp. on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.
- M. Sosnowski et al. The performance of post-quantum tls 1.3. In *International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT 2023, page 19–27, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400704079. doi: 10.1145/3624354.3630585. URL <https://doi.org/10.1145/3624354.3630585>.
- M. T. d. Souza, M. A. Souza, and H. C. Freitas. Quantum machine learning with enhanced autoencoders for intrusion detection. In *Anais do XXVI Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 217–228, Porto Alegre, RS, Brasil, 2025. SBC. doi: 10.5753/sscad.2025.16683. URL <https://sol.sbc.org.br/index.php/sscad/article/view/37900>.