IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Manuscript prepared July 2026.

Hig: Engineering a Project-Aware Incremental Archiving Framework for Repeated Developer Workflows

YIKE WANG[1]
[1]Vtslx AI Research, Haokir Inc., Colorado Springs, CO 80918 USA (e-mail: yikewang@research.hydite.com)
Corresponding author: Yike Wang (e-mail: yikewang@research.hydite.com).

No funding was received for this research. ORCID: 0009-0009-1832-5421.

ABSTRACT Developer projects are repeatedly archived during local builds, handoffs, checkpoints, and tool-assisted workflows. Conventional archive utilities optimize an individual invocation, whereas repeated project archiving must also manage change discovery, secure reuse, bounded memory, and variable local storage behavior. This paper presents Hig, a project-aware incremental archiving framework whose current first-major-release line is v1.9.7. Hig combines an HIGV2 archive surface, project snapshots, a watcher-assisted change journal, reusable prepared objects, bounded-memory pipeline staging, and standard cryptographic primitives. It does not introduce a new compression, encryption, hash, or cryptographic algorithm. Instead, its contribution is the engineering framework that coordinates established techniques for recurring developer workflows. The paper reports an evidence-based evolution from v1.0.1 through v1.9.7, separating controlled comparisons from observations collected under different I/O conditions. In a controlled v1.9.6/v1.9.7 cold-path study, v1.9.7 reduced source-read worker time by 75.8% while preserving archive size and tested cross-version extraction compatibility. A 15,330-file large-project comparison is reported as contextual evidence only because the archived record does not preserve all baseline tool settings. Real-project results also reveal non-stationary storage effects that preclude a blanket throughput claim. The paper documents these limits, the disclosure boundary, benchmark provenance, and an engineering-validation protocol for future releases.

INDEX TERMS Archiving, developer tools, incremental systems, project-aware storage, reproducible benchmarking, secure systems.

I. INTRODUCTION

DEVELOPER projects are archived repeatedly: before experimentation, during local automation, while moving work between machines, and as a compact project-state handoff. In this setting, the cost of a pack operation is not defined solely by compression throughput. It also includes determining what changed, deciding which previously prepared objects may be reused, allocating bounded memory, preserving a stable archive surface, and writing to a storage device whose latency may vary substantially.

Hig is a Rust implementation intended for these recurring workflows. Its first release line evolved from a file-level cached archiver in v1.0.1 to the HIGV2 format and, subsequently, project snapshots, daemon-backed coordination, watcher-assisted invalidation, and cold-path reuse refinements in v1.9.7 [18]. The work is best understood as a systems and algorithmic framework. Hig uses Zstandard compression [7], BLAKE3 hashing [10], Argon2id password derivation [8], and ChaCha20-Poly1305 authenticated encryption [9]; it does not claim a new primitive in any of these categories.

The paper makes five contributions.

1) It defines a public, non-sensitive functional model for project-aware incremental archiving and states the boundary between the published system description and proprietary implementation details.
2) It reconstructs the Hig evolution across v1.0.1–v1.9.7 from release artifacts, including both positive results and failed or inconclusive optimization outcomes.
3) It evaluates archive size, warm project workflows, controlled cold-path behavior, memory policy, correctness, and compatibility using recorded benchmark artifacts.
4) It adds an engineering verification path that connects build gates, unit tests, cross-version extraction, digest

checks, and benchmark qualification to the claims made in the paper.
5) It provides a measurement discipline for reporting future HIG releases without conflating distinct corpora, security modes, or storage conditions.

The rest of this paper reviews related systems in Section II, describes the framework in Section III, explains the empirical method and engineering verification path in Section IV, reports results in Section V, discusses limitations in Section VI, and concludes in Section VII.

II. RELATED WORK

Incremental transfer and storage systems have long exploited similarities between successive data states. The rsync algorithm uses rolling checksums to identify matching blocks across endpoints [1]. LBFS applies content-defined chunking to exploit similarity across files [2]; Venti uses content-addressed archival storage [3]. These works establish the value of reuse, but their primary settings differ from a local developer-oriented archive workflow that must integrate filesystem change observation and an archive writer.

Deduplicating backup systems, including Data Domain [4] and foundational chunking studies [5], show the trade-off between chunk granularity, metadata cost, and reuse. Hig adopts the general insight that a single block treatment is not appropriate for all files. Its published model therefore contains small-object aggregation, large-object chunk handling, and independent payload preparation, while intentionally omitting the private thresholds and internal cache schema.

Archive format and compression design have different priorities. The ZIP specification [6] remains a ubiquitous interoperability baseline, while Zstandard specifies a modern compressed-data format [7]. Hig retains a stable HIGV2 surface during the v1.9.7 cold-path work, thereby treating performance refinement as separable from format migration.

Modern tools further clarify the design boundary. Borg and Restic are repository-oriented backup systems with content reuse, encryption, and retention-oriented workflows [13], [14]. Git represents project history through content-addressed objects [15], and Bazel remote caching reuses action outputs rather than general archive payloads [16]. Hig is not presented as a replacement for these systems. Its scope is a portable archive artifact for a repeatedly changing local project, with a project-state layer intended to reduce redundant preparation before that artifact is emitted. No experiment in this paper claims a head-to-head comparison with Borg, Restic, Git, or Bazel; such a comparison requires workload-specific semantics and fairness controls beyond the preserved artifact set.

Security-sensitive archive tools must make explicit use of established primitives and threat assumptions. Argon2id is specified for password hashing [8], ChaCha20-Poly1305 provides authenticated encryption [9], and BLAKE3 supplies a fast cryptographic hash function [10]. The framework described here composes such primitives; it does not modify their security definitions. This distinction matters because performance measurements for encrypted and unencrypted workflows are not directly interchangeable.

III. FRAMEWORK DESIGN AND DISCLOSURE BOUNDARY

A. DESIGN GOALS

Hig targets five properties: (i) compact archives for mixed developer trees, (ii) efficient repeated pack operations, (iii) integrity-aware secure defaults for password archives, (iv) bounded resource behavior, and (v) stable extraction compatibility during optimization. The implementation offers balanced and fastest modes, with explicit security and trust choices rather than silently weakening the default password path.

B. PUBLIC FUNCTIONAL ARCHITECTURE

Fig. 1 presents the public functional view. A client invokes a pack operation directly or through a local coordination service. In project mode, a watcher and change journal maintain a project snapshot. The scan stage validates current file state and produces work descriptions. A planner groups or chunks work at a high level, the bounded-memory stage prepares payloads, and the writer emits the archive and protected metadata/payloads when password encryption is selected.

This structure supports a task ordering policy that can prioritize smaller work while maintaining deterministic output ordering, and a buffer pool that bounds reusable working storage. These are scheduling and resource-management mechanisms, not novel compression or cryptographic algorithms.

C. EVOLUTION OF THE FRAMEWORK

Fig. 2 summarizes the major evidence-backed stages. Early v1.0.1 artifacts demonstrate file-level block reuse but also expose poor small-file behavior. v1.2.0 introduces HIGV2 batch and chunk representations. The v1.8 series adds mature daemon/session and release-gate instrumentation. The v1.9 series focuses on watcher-assisted project mode, write-path telemetry, adaptive memory behavior, and reduction of duplicate cold-path reads.

D. DISCLOSURE BOUNDARY

The paper intentionally reports externally observable behavior, compatibility, and experimental measurements. It does not disclose internal cache schemas, exact block-selection thresholds, raw key-handling details, storage paths, proprietary scheduling heuristics, or deployment hardening procedures. This boundary is compatible with reproducible evaluation: commands, corpus characteristics, versions, modes, output metrics, and checksums are reported without publishing internal secrets.

IV. EVALUATION METHODOLOGY

A. ARTIFACT SET AND VERSION SCOPE

The analysis uses the project release artifact set from v1.0.1 through v1.9.7 [17]. It includes synthetic source, repetitive, random, and small-file corpora; release-gated source-tree

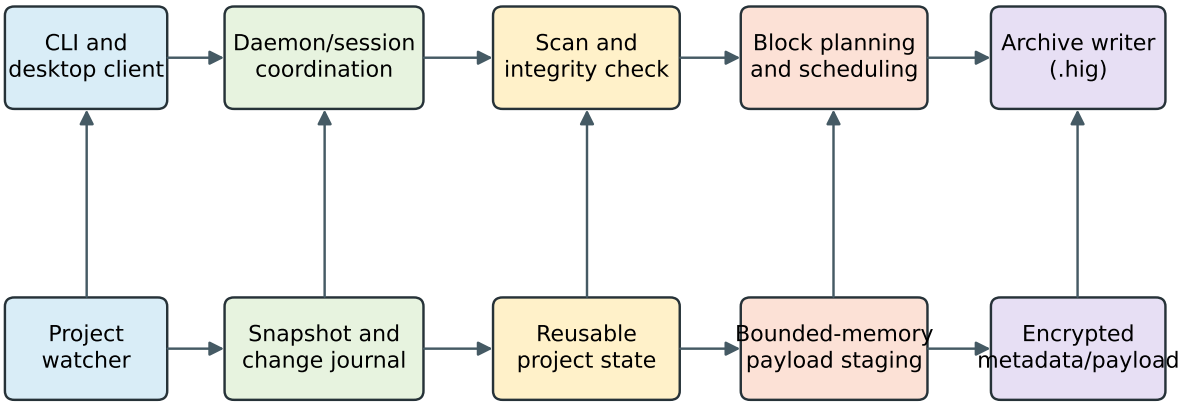

FIGURE 1. **Public functional architecture of Hig. The diagram intentionally omits cache layouts, numeric policies, key-handling details, and implementation-specific heuristics.**

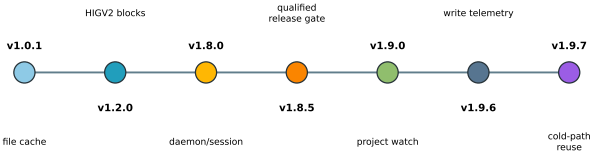

FIGURE 2. **Evidence-based Hig evolution, v1.0.1 through v1.9.7. The labels identify publicly observable capability milestones, not a claim of monotonic performance across incomparable environments.**

TABLE 1. **Evaluation families and interpretation rules.**

Family	Primary evidence	Valid interpretation
v1.0–v1.2 synthetic	Source, repetitive, random, small files	Functional evolution and edge cases only
v1.6–v1.8 release gates	Qualified and unqualified local volumes	Same-run tool comparisons; qualification status retained
v1.9 project watch	15,330-file large developer project corpus	Warm workflow behavior and stage hotspots
v1.9.7 cold path	Interleaved v1.9.6/v1.9.7 runs	Causal comparison for retained-read optimization
v1.9.7 real IDE	17,583 files, 505.9 MB	Memory behavior and qualitative I/O sensitivity

benchmarks; a 15,330-file large developer project corpus; a 17,583-file real IDE corpus; and controlled v1.9.6/v1.9.7 cold-path corpora. The suite is longitudinal rather than a single universal benchmark. Therefore, numbers are compared directly only when corpus, command mode, version pairing, and environment conditions support the comparison.

B. METRICS

The study records archive bytes, core duration, CLI wall time where available, stage timings, cache reuse, watcher status, memory staging, and source-to-extracted-tree digest equality. Archive size is a direct byte count. Duration is reported as recorded by the associated artifact, typically a median when repeated samples exist. This follows reproducible-research and rigorous-performance-evaluation principles [11], [12]. A faster result is not inferred from a single sample when the artifact reports non-stationary I/O.

For each archive, the size ratio is

$$R_s = \frac{B_{archive}}{B_{input}}, \tag{1}$$

C. BENCHMARK PROVENANCE AND REPRODUCTION ENVELOPE

The retained artifacts preserve corpus counts and bytes, Hig version, cache state, daemon mode, encryption mode, memory policy, repeat count, and selected command shapes. For the controlled cold-path study, they also preserve the input digest and interleaving rule. However, the historical v1.9.6 large-project comparison does not consistently preserve host model, operating-system build, filesystem, baseline utility version, compression level, or all baseline command flags. Table 2 makes this distinction explicit. Consequently, the controlled v1.9.6/v1.9.7 cold-path pair is the paper's causal performance evidence; the historical utility comparison is retained as a transparent contextual observation.

Future Hig evaluations should publish a machine-readable host manifest, exact command lines and baseline versions, generated public corpora with deterministic seeds, raw rep-

TABLE 2. Benchmark provenance and reproducibility envelope. "Recorded" means retained by the cited release artifact, not necessarily publicly distributable.

Study	Recorded controls	Missing or restricted information	Interpretation in this paper
v1.9.6 Project-A utility comparison	File count, input bytes, archive bytes, run times, and environment qualification	Proprietary corpus; baseline utility versions, all flags, host and filesystem metadata not retained consistently	Contextual size/time observation; not a causal ranking
v1.9.6 project-watch workflow	Project state, edit scenarios, watcher counters, stage telemetry, and warm-path target	Proprietary corpus and complete host metadata	Measured workflow behavior and bottleneck attribution
v1.9.7 cold-path pair	Corpus digest and composition, fresh caches, output isolation, balanced mode, daemon off, encryption state, run counts, and interleaving	Operating-system page cache not flushed; complete host manifest absent	Primary controlled comparison of retained-read work
v1.9.7 real-project memory study	File/byte counts, mode, fresh cache, daemon/project state, run count, spool and memory counters	Proprietary corpus; non-stationary storage latency; complete host manifest absent	Resource-policy trade-off, not throughput ranking

etitions, and confidence intervals. Until those materials are released, the protocol is internally auditable rather than externally fully reproducible. where B denotes bytes. For paired controlled experiments, the relative change in a metric x is reported as

$$\Delta_x = \frac{x_{v1.9.7} - x_{v1.9.6}}{x_{v1.9.6}} \times 100\%. \quad (2)$$

D. CORRECTNESS AND COMPATIBILITY

Correctness requires successful extraction and equality of the restored tree digest with the input digest. The v1.9.7 cold-path experiments additionally use bidirectional cross-version extraction: v1.9.6 archives are extracted by v1.9.7 and v1.9.7 archives are extracted by v1.9.6. This test detects an unintended archive-surface change even when a same-version round trip would succeed.

E. ENGINEERING VERIFICATION PATH

The engineering verification path is intentionally broader than benchmark timing. A result is treated as publishable only when the associated implementation state passes the relevant build and correctness gates, the archive can be unpacked, and the measurement record states its environment qualification. Table 3 summarizes the verification layers used by the release artifacts.

F. MULTIDIMENSIONAL EVIDENCE MAP

The evaluation deliberately separates evidence dimensions instead of reducing the system to a single speed score. Fig. 3 maps benchmark families to evidence types. ''Ctrl.'' means a controlled or compatibility-validated result; ''Meas.'' means directly measured telemetry; ''Obs.'' means an observation useful for interpretation but insufficient for a causal performance claim. This prevents a common reporting error: using a strong result in one dimension, such as source-read reduction, to imply unmeasured strength in another dimension, such as stable end-to-end wall time on every storage device.

V. RESULTS

A. LONGITUDINAL FINDINGS

The earliest v1.0.1 artifacts establish why a project-aware framework is needed. On 500 tiny files totaling 18 kB, the first archive was 142,524 B, an expansion of 691.8%. Conversely, repetitive data compressed effectively. These findings motivated aggregation in HIGV2. In v1.2.0, HIGV2 batch mode archived a 133,195-B source corpus to 28,250 B, compared with 31,238 B without batching and 32,967 B in the legacy one-file-per-block format. The result is a format-level observation on that corpus, not a universal compression ranking.

The v1.8 release artifacts show a mature measurement posture: repeat sampling, volume qualification, cache and stage telemetry, and explicit security-mode labels. In qualified v1.8.5 large-project measurements, the balanced secure daemon pack-core median was 377.56 ms and zip was 3,750.94 ms on the recorded workload. Later v1.9 artifacts add project-watch scenarios and explicitly retain failed gates when output I/O prevents meeting the target.

B. LARGE-PROJECT ARCHIVE COMPARISON

The v1.9.6 Project-A experiment uses an anonymized large developer project corpus with 15,330 files and 198,974,616 input bytes. Fig. 4 reports the retained first-pack record alongside common utilities. The HIG archive was 57.11 MB, while the recorded zip and tar.gz outputs were 67.75 MB and 61.33 MB. The recorded first-pack times were 1.18 s for Hig, 4.38 s for zip, 10.89 s for tar.gz, and 8.50 s for tar.zst. These values are not treated as a fair causal ranking: the volume was unqualified and the artifact does not preserve baseline utility versions, levels, or complete command flags. They are retained to document the historical workload and motivate the controlled study, not to establish general superiority.

C. PROJECT-MODE WORKFLOWS

For v1.9.6, project snapshot reuse recorded 15,330 hash reuses and no watcher overflows. The project bootstrap pack took 1,430.03 ms; a single-edit pack took 187.42 ms; a five-edit pack took 174.41 ms; and a 1,000-event burst pack took 293.77 ms (Fig. 5). The project warm median was 169.99

TABLE 3. Engineering verification path for the reported claims.

Layer	Evidence source	Claim supported	Boundary
Static quality	Formatting, lints, release builds, and package checks	Source state is buildable and mechanically consistent	Does not prove performance
Core behavior	Unit and integration tests for packing, unpacking, writer behavior, sessions, and project state	Functional regressions are screened before benchmarking	Test coverage is finite
Archive correctness	Pack/unpack smoke tests and tree digest equality	Archives restore the measured corpus exactly	Limited to measured corpora
Compatibility	Cross-version extraction between v1.9.6 and v1.9.7 in the cold-path study	The tested HIGV2 surface remained compatible	Not a complete format proof
Performance telemetry	Stage timings, cache counts, memory bytes, and write-profile counters	Bottleneck attribution can be separated by stage	Sensitive to I/O state
Environment qualification	Copy-throughput checks and explicit qualified/unqualified status	Absolute timing claims are scoped to the observed environment	Does not normalize hardware

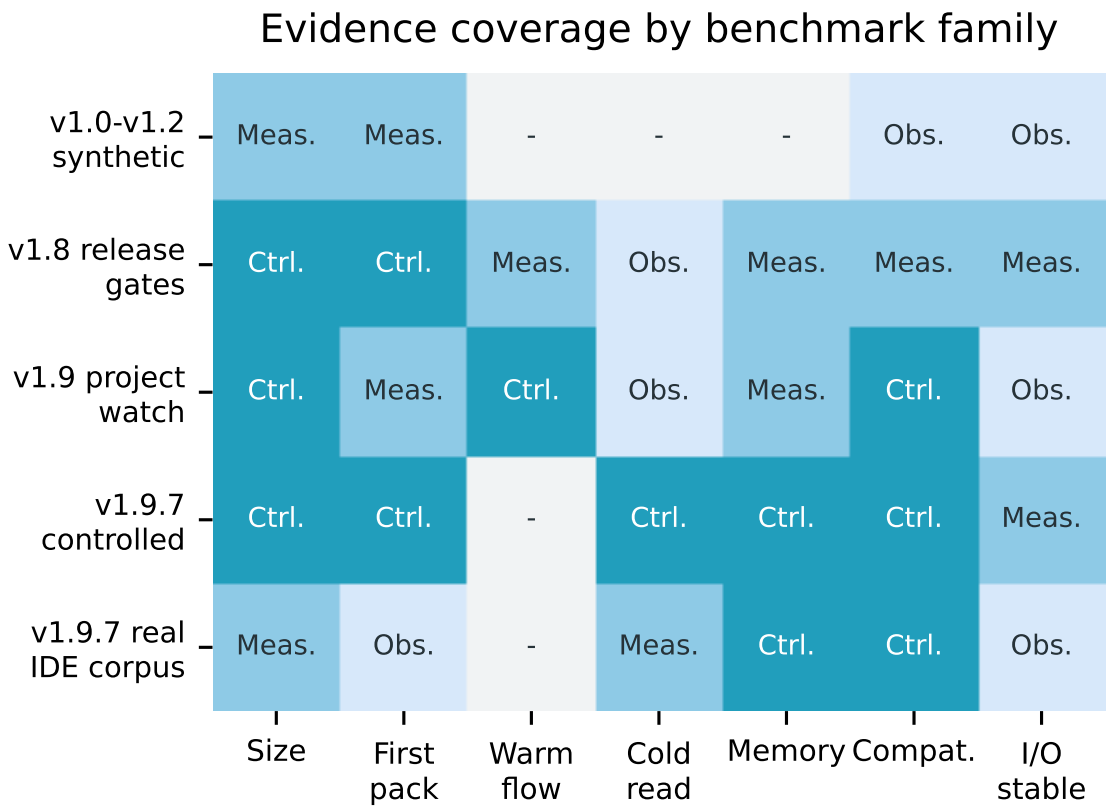

FIGURE 3. Multidimensional evidence coverage by benchmark family. The matrix distinguishes controlled validation from measured and observational evidence, keeping engineering claims scoped to their supporting data.

TABLE 4. v1.9.6 Project-A first-pack results.

Tool	Time (s)	Size (MB)	R_s
HIG project pack	1.183	57.110	28.70%
zip	4.384	67.749	34.05%
tar.gz	10.887	61.333	30.82%
tar.zst	8.504	64.936	32.64%

ms, missing the 150-ms target. Stage telemetry identifies output write (105.18 ms median), output flush (38.29 ms), and cryptographic work (20.98 ms) as the largest observed contributions. Reporting the miss is essential: a project-aware system can eliminate data-read and compression work for unchanged payloads while still being constrained by output and filesystem behavior.

D. CONTROLLED V1.9.7 COLD-PATH COMPARISON

The controlled cold-path experiment uses a 6,154-file corpus of approximately 291 MiB, new caches, balanced compression, daemon disabled, and interleaved runs. The unencrypted median source-read worker time decreased from 810.5 ms in v1.9.6 to 196.2 ms in v1.9.7 ($\Delta = -75.8\%$). Core duration

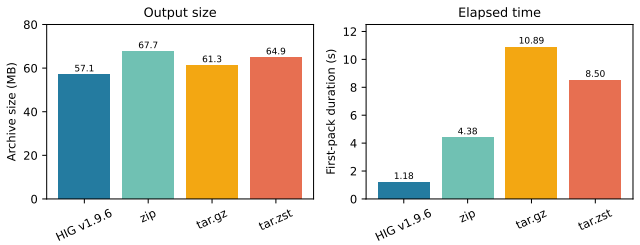

FIGURE 4. v1.9.6 Project-A first-pack record. The 15,330-file large developer project corpus was measured on an environment marked unqualified by the release gate. Missing baseline configuration metadata makes this a contextual observation rather than a causal tool comparison.

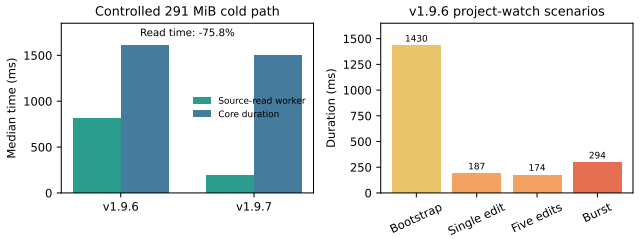

FIGURE 5. Left: interleaved controlled v1.9.6/v1.9.7 cold-path medians. Right: v1.9.6 project-watch scenarios. The panels represent different corpora and should not be compared as one benchmark.

decreased from 1,605.5 to 1,495.5 ms ($\Delta = -6.9\%$), while the 119.60-MB archive size showed no material change. In password-encrypted runs, read time decreased by 74.5%, but end-to-end duration was neutral within output-I/O variance. These results support the narrow claim that v1.9.7 reduces duplicate source-read work for its retained window; they do not establish a universal first-pack speedup.

E. MEMORY POLICY AND REAL IDE CORPUS

The 17,583-file real IDE corpus is useful for resource-policy evaluation but not for blanket speed ranking. The default v1.9.7 adaptive mode used no payload spool bytes and recorded 289.56 MB peak pipeline memory, compared with 125.83 MB in an explicit fixed low-memory spool configuration. The low-memory configuration moved 180.51 MB through a same-volume spool. Absolute durations were collected under severe non-stationary storage latency; the reports correctly reject attributing the apparent 7.24× duration difference solely to memory policy. A subsequent hot-raw experiment eliminated block-preparation source rereads for the full 505.9-MB corpus but increased reported peak pipeline memory to 795.47 MB. The appropriate conclusion is a transparent time-memory trade-off, not a default throughput claim.

F. WRITE-PATH DIAGNOSTICS

The v1.9.7 write-path profile is important because it explains why read-path improvements do not automatically become end-to-end wall-time improvements. On a 292-MB benchmark corpus with 163.98 MB of archive payload bytes, the median total write stage was 658 ms. Payload writing contributed 641 ms, approximately 97% of the measured write stage. Manifest writing, header writing, temporary-file creation, preallocation, and rename were each negligible by comparison. Fig. 7 therefore identifies a concrete next engineering target: reducing the number or cost of buffered memory-payload writes. This diagnostic also protects the paper from overclaiming v1.9.7 as a complete performance endpoint; the current release line still contains visible write-path work.

TABLE 5. Controlled cold-path v1.9.6/v1.9.7 results.

Metric	v1.9.6	v1.9.7	Change
Core duration (ms)	1,605.5	1,495.5	-6.9%
Source-read worker (ms)	810.5	196.2	-75.8%
Block preparation (ms)	865.0	784.0	-9.4%
Retained raw bytes	0	64 MiB	+64 MiB
Archive size (MB)	119.60	119.60	no material change

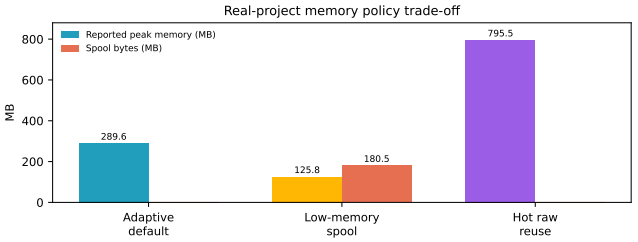

FIGURE 6. Memory-policy trade-off on the 17,583-file real-project corpus. Adaptive mode avoids payload spooling, explicit low-memory mode lowers peak memory by using a same-volume spool, and hot-raw reuse removes repeated source reads at the cost of higher retained memory.

G. COMPATIBILITY

Both secure cold-path archives extracted correctly with both v1.9.6 and v1.9.7. The real IDE adaptive archive restored all 17,583 files and all 505,906,599 bytes. Together with unchanged HIGV2 format claims in the v1.9.7 artifacts, these checks support compatibility for the tested cases only; they are not a formal proof over all possible archives.

VI. DISCUSSION AND THREATS TO VALIDITY

A. WHAT THE EVIDENCE SUPPORTS

The longitudinal record supports the central framework claim: archive preparation benefits from coordinating change awareness, reuse, scheduling, memory staging, and stable output behavior. It also supports specific narrow claims: HIGV2 aggregation reduced overhead on small source-like files in its recorded v1.2.0 corpus; project mode reuses unchanged project state; and v1.9.7 reduces duplicate reads in controlled retained-window tests.

B. WHAT THE EVIDENCE DOES NOT SUPPORT

The data do not prove a new compression or cryptographic algorithm, a universal throughput advantage, or a monotonic speed increase across versions. Early synthetic tests, qualified release gates, unqualified volumes, anonymized large-project workloads, and real IDE workloads use different corpora and

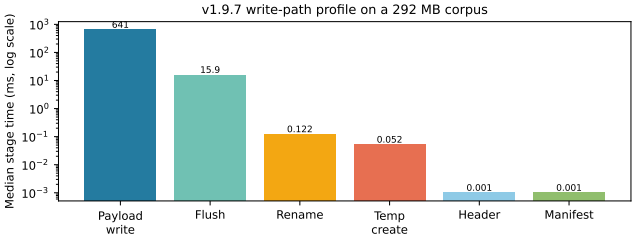

FIGURE 7. Median v1.9.7 write-path profile on a 292-MB corpus. Payload writing dominates the stage, so read-path optimizations can be masked by final payload emission and flush behavior.

environmental states. The real IDE report explicitly records large storage-latency variance. The Project-A utility record also lacks complete baseline configuration metadata. It would be methodologically incorrect to collapse those measurements into a single cross-version speed curve or to use the utility record as a claim of general superiority.

The current compatibility result covers two versions and the measured corpora only. It does not establish behavior for symbolic links, sparse files, extended attributes, access-control lists, malformed archives, path-normalization edge cases, or all cross-platform filesystem combinations. Likewise, the security discussion identifies standard primitives but does not constitute a third-party security audit or a complete threat model. These exclusions are material limitations, not omitted footnotes.

C. MATURITY AND FUTURE WORK

Version v1.9.7 is the current first-major-release line, not a completed feature set. Future work includes stable-NVMe repetition of real-project benchmarks, broader platform validation, improved output-write telemetry and coalescing, adaptive large-file policies, formalized threat modeling, and more extensive compatibility test matrices. New algorithm research would be justified only after these systems-level bottlenecks are isolated and the existing framework has reached its practical limit.

VII. CONCLUSION

This paper presents Hig as a project-aware incremental archiving framework, not as a claim of new cryptographic or compression primitives. The v1.0.1–v1.9.7 artifact record shows the progression from file-level reuse to HIGV2 block planning, local coordination, watcher-assisted project state, and controlled cold-path reuse. The strongest quantitative conclusion is deliberately narrow: v1.9.7 reduced controlled source-read worker time by 75.8% without changing measured archive size or tested cross-version extraction behavior. Other measurements demonstrate compact project archives and useful warm workflows, while also exposing output-I/O and memory trade-offs. By preserving negative results and environment qualification, the paper provides a defensible baseline for future Hig releases and for eventual investigation of genuinely new algorithms.

DATA AVAILABILITY STATEMENT

The proprietary project trees used in the Project-A and real IDE studies cannot be distributed. Aggregate measurements, corpus counts, corpus digests where recorded, command shapes, and validation outcomes are reported in this article. The release artifacts used to prepare the manuscript are retained by the author; non-sensitive supporting material may be requested from the corresponding author, subject to confidentiality and availability.

CONFLICT OF INTEREST

The author declares no conflicts of interest.

REFERENCES

[1] A. Tridgell and P. Mackerras, "The rsync algorithm," Australian National University, Tech. Rep. TR-CS-96-05, 1996.
[2] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proc. ACM SOSP*, 2001, pp. 174–187.
[3] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proc. USENIX FAST*, 2002, pp. 89–101.
[4] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the Data Domain deduplication file system," in *Proc. USENIX FAST*, 2008, pp. 1–14.
[5] A. E. Broder, "Some applications of Rabin's fingerprinting method," in *Sequences II*, 1993, pp. 143–152.
[6] P. Katz, ".ZIP file format specification," PKWARE, Inc., APPNOTE, 2024. [Online]. Available: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT
[7] Y. Collet and M. Kucherawy, "Zstandard compression and the application/zstd media type," RFC 8878, Feb. 2021.
[8] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2 memory-hard function for password hashing and proof-of-work applications," RFC 9106, Sep. 2021.
[9] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," RFC 8439, Jun. 2018.
[10] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and J. W. O'Connor, "BLAKE3: One function, fast everywhere," 2020. [Online]. Available: https://github.com/BLAKE3-team/BLAKE3-specs
[11] V. Stodden, F. Leisch, and R. D. Peng, Eds., *Implementing Reproducible Research*. Boca Raton, FL, USA: CRC Press, 2014.
[12] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *Proc. ACM OOPSLA*, 2007, pp. 57–76.
[13] The Borg Collective, "Borg documentation," 2026. [Online]. Available: https://borgbackup.readthedocs.io/en/stable/
[14] Restic Contributors, "Restic documentation," 2026. [Online]. Available: https://restic.readthedocs.io/en/stable/
[15] S. Chacon and B. Straub, "Git internals: Git objects," *Pro Git*. [Online]. Available: https://git-scm.com/book/en/v2/Git-Internals-Git-Objects
[16] Bazel Authors, "Remote caching," 2026. [Online]. Available: https://bazel.build/remote/caching
[17] Y. Wang, "Hig versioned benchmark and release artifacts, v1.0.1–v1.9.7," Vtslx AI Research, Haokir Inc., internal technical artifact collection, 2026.
[18] Y. Wang, "Hig source implementation and HIGV2 compatibility tests," Vtslx AI Research, Haokir Inc., software repository, version 1.9.7, 2026.