PREPRINT · GW-2026-0004

Cancellation-Safe Native Loop Lowering in the Gray Runtime

A 25.918x median end-to-end speedup over Gray VM execution with a stop-token safepoint on every generated backedge

Author: Annabella Proctor
ORCID: https://orcid.org/0009-0000-0799-1757
Affiliation: Grayworth
Publisher: Grayworth Technical Publications
Manuscript date: July 27, 2026
Version: 1.0
DOI: https://doi.org/10.5281/zenodo.21633227
ISBN-13: 9798191411620

Abstract

This paper evaluates bounded x86-64 native lowering for an eligible typed loop in the Gray language runtime. Historical profiling of Gray's sustained interpreter path recorded approximately 6.91 billion retired instructions and 1.60 billion branches with only 78,503 branch misses, indicating that dispatch and instruction volume, rather than branch-prediction failure, dominated the workload. The intervention lowers a supported loop to native code while preserving Gray's runtime ABI, source-function identity, structured exits, exact integer result, and cooperative cancellation. Seven interleaved, fresh-process, CPU-pinned measurements of a 100-million-iteration sum produced a native minimum, median, and maximum of 201.846, 210.511, and 259.657 ms. The same function with JIT disabled produced 5,336.574, 5,456.099, and 5,618.863 ms, yielding a 25.918x median end-to-end speedup. Both policies returned exactly 4,999,999,950,000,000. A separate 9-quintillion-iteration diagnostic requested cancellation after 20 ms and required CancelledError; Live JIT, Live VM, Static JIT, Static VM, AddressSanitizer, and UndefinedBehaviorSanitizer gates passed. The generated loop performs a stop-token safepoint at every backedge, so the measured native path includes the cooperative-interruption mechanism rather than benchmarking a semantics-stripped loop. The result demonstrates that Gray can remove the dominant bytecode-dispatch cost from an eligible hot region while retaining the runtime behavior needed for safe server operation.

Keywords: Gray; JIT compilation; x86-64; bytecode interpreter; native lowering; safepoint; cooperative cancellation

1. Introduction

An interpreter gives a young language an unusually productive execution substrate. The compiler can emit a compact instruction form, the runtime can preserve rich source identity, and semantics can evolve without first building a complete optimizing backend. The cost arrives in sustained loops: each source operation becomes one or more bytecode dispatches, operand decodes, value checks, and stack or frame transitions. Once the surrounding program is stable, repeated execution pays that machinery again on every iteration.

Gray's historical sustained sum made the cost visible. Linux perf recorded about 6.91 billion retired instructions and 1.60 billion branches, but only 78,503 branch misses. The profile did not point primarily to a pathological branch predictor. It pointed to the volume of work required to interpret a simple typed loop. A benchmark-specific fused bytecode could reduce one diagnostic, but it would leave the general dispatch boundary intact. The architectural response was bounded native lowering: compile a supported hot region to machine code, enter it through the runtime ABI, and return through explicit result states.

Server-language optimization cannot stop at arithmetic speed. A long native loop that ignores cancellation can pin an application during shutdown, request abort, Adaptive replacement, or operator intervention. Gray therefore places a stop-token check on every generated loop backedge. The experiment measures the loop with that check present. Seven interleaved samples show a 210.511 ms native median versus 5,456.099 ms in the VM, a 25.918x end-to-end speedup, while a separate effectively unbounded loop still observes cancellation after a 20 ms request.

The result is both narrow and large. It does not claim that every Gray function is natively lowered or that every application becomes 25.918 times faster. It establishes the critical mechanism: for an eligible loop, Gray can eliminate the dominant interpreter cost without abandoning source identity, structured exits, execution-policy consistency, or cooperative control.

2. Research questions

The first question is whether bytecode dispatch is a dominant cost in Gray's eligible sustained integer loop. Historical whole-process timings and hardware-counter evidence provide the diagnosis: the VM performs billions of instructions and branches despite very

few branch misses. The intervention must produce a change commensurate with removing that work, not a marginal improvement attributable to measurement noise.

The second question is whether the native region computes exactly the same result as the VM across Live and Static policies. The benchmark sums integers from zero through 99,999,999 and must print 4,999,999,950,000,000. A timing is rejected if output differs. Structured loop exits and return state must reach the owning VM through declared result flags rather than jumping around runtime ownership.

The third question is whether cooperative cancellation survives native lowering. A generated loop can be fast precisely because it remains in machine code for a long time; that same property can make it unresponsive. The native backedge therefore executes a stop-token safepoint on every iteration. A separate 9,000,000,000,000,000,000-iteration function is canceled after 20 ms and must surface CancelledError rather than complete, hang, crash, or return an ordinary value.

The final question is whether the measured result is stable enough to support a speedup claim. JIT-enabled and JIT-disabled runs are interleaved, each in a fresh process pinned to one CPU, after correctness warmups. Minimum, median, and maximum are retained for seven accepted rounds per policy. The median ratio is calculated from those complete sets.

3. Contributions

The compiler contribution is a bounded x86-64 lowering path for a typed loop shape with explicit eligibility checks. The backend maps supported integer operations and loop control to native instructions, preserves the runtime calling convention, associates the region with its Gray source function, and returns success, structured exit, or cancellation through an explicit contract. Unsupported values or control forms remain in the VM rather than being guessed into machine code.

The runtime contribution is cancellation-safe execution. Every generated backedge checks the owning stop token. The check is part of the measured loop body, making the 25.918x result an honest systems result rather than a comparison between a fully governed interpreter and an uninterruptible native kernel. The separate cancellation diagnostic, sanitizer runs, and four-mode semantic matrix verify the boundary independently from the finite timing loop.

The empirical contribution is seven interleaved whole-process samples for both execution policies, exact output validation, immutable runtime and fixture hashes, and an explanatory connection to the prior hardware-counter profile. Native measurements ranged from 201.846 to 259.657 ms with a 210.511 ms median. VM measurements ranged from 5,336.574 to 5,618.863 ms with a 5,456.099 ms median. The ratio of medians is 25.918.

The methodological contribution is a model for introducing optimization into a server runtime without defining semantics downward. Performance acceptance includes cancellation, source and ABI ownership, execution-policy agreement, and sanitizers. The backend earns a larger eligibility surface only after those properties remain intact.

4. Background and related work

Bytecode interpreters repeatedly fetch, decode, and dispatch virtual instructions. Ertl and Gregg showed that efficient interpreters execute unusually high proportions of indirect branches and that dispatch organization interacts strongly with hardware prediction. Gray's profile adds an implementation-specific observation: in its sustained sum, absolute branch and instruction volume was enormous while branch misses were low. Improving prediction would not erase billions of dispatch-related operations; entering a native region can.

Just-in-time compilation spans a wide design space, from tracing systems and tiered optimizing compilers to baseline translation. Gray's first native loop path is deliberately bounded. It does not speculate over arbitrary dynamic values, perform global optimization, or deoptimize a large compiled graph. Eligibility is proven from the typed function and supported control shape; unsupported code continues in the mature VM. This limits initial complexity while targeting the profile's dominant repeated cost.

Safepoints are established compiler/runtime coordination points. LLVM's statepoint documentation discusses locations where machine state can be made interpretable for garbage collection and notes loop backedges as a standard placement site for polls. Gray's stop-token poll serves cooperative cancellation rather than relocating collection, but the engineering principle is related: generated code must periodically re-enter a state where the runtime can impose a global or owner-specific control decision.

Established native compilers provide useful performance context but are not the measured baseline in this diagnostic. Historical container records show C++ and Go completing a different 100-million-iteration sustained sum in tens of milliseconds, while the older

Gray interpreter required hundreds of milliseconds in a differently shaped workload. Those figures motivated backend work; they are not mixed into the 25.918x ratio. The reported comparison is Gray native versus Gray VM for the exact same current function, binary, CPU placement, and output.

5. Gray system architecture

Gray source is compiled into a runtime representation used by Live and Static execution. The VM owns values, call frames, errors, stop tokens, source locations, and interactions with integrated server facilities. Static artifacts can embed bytecode and eligible native machine code so deployment does not require invoking a startup compiler for those regions. The native backend must therefore coexist with the VM rather than replacing its ownership model wholesale.

Eligibility is intentionally conservative. The compiler recognizes a typed loop whose values and operations have supported native representations and whose control exits can be mapped to the backend result contract. It emits x86-64 code under the runtime ABI, records the source-function association, and installs entry metadata. If any required property cannot be established, execution remains in bytecode. A fallback is a correctness feature, not a benchmark failure.

At entry, the VM passes the native region its arguments and runtime context. The generated code performs the loop arithmetic in registers and native control flow. At each backedge it reads the stop token and branches to a cancellation return when requested. Normal completion returns the result and success state; structured exits use their declared state. The owning VM converts that state into the same language-level behavior used by interpreted execution.

The per-backedge check is stronger than an occasional timer poll. Its overhead is paid on every iteration in the reported 100-million-iteration result, while its latency bound is tied to the duration of one generated loop body plus runtime handling. That choice prioritizes predictable cancellation for server workloads. Later backends may safely coarsen checks only with a separately defined responsiveness contract.

Live and Static use the same semantic boundary even though code acquisition differs. Live may compile an eligible region during program execution; Static can relocate embedded native material. The diagnostic's four-mode gate ensures that enabling or disabling

native execution does not alter the observable sum or cancellation result in either product policy.

6. Experimental design

The finite benchmark consists of one typed Gray function that sums integers from zero through 99,999,999. Each accepted process must print exactly 4,999,999,950,000,000. One policy permits native lowering. The comparison policy sets GRAY_JIT=0, forcing the same function through VM execution. Processes are pinned to CPU 1, and policy samples are interleaved after correctness warmups so a monotonic host drift is less likely to favor one side.

Seven samples are retained for each policy. Timing is whole-process for the defined diagnostic rather than an internal cycle count, so backend selection, entry, loop execution, and return behavior remain in scope. The median is the primary statistic; minimum and maximum show spread. Speedup is VM median divided by native median: 5,456.099 / 210.511 = 25.918 after rounding to three decimals.

The cancellation fixture uses a loop bound of 9,000,000,000,000,000,000, chosen so ordinary completion is not a plausible explanation. A spawned Gray VM begins the loop, the owner requests cancellation after 20 ms, and the program must produce CancelledError. The gate runs under Live JIT, Live VM, Static JIT, and Static VM. AddressSanitizer and UndefinedBehaviorSanitizer builds exercise the native ownership and exit path.

Historical perf data is explanatory evidence, not part of the timing ratio. The approximately 6.91 billion retired instructions, 1.60 billion branches, and 78,503 branch misses came from the prior sustained VM baseline. It supports the hypothesis that eliminating interpreted instruction volume would have a large effect. The current interleaved experiment tests the intervention directly.

The scope is one eligible arithmetic loop on x86-64. The strong claim is that native lowering removed the dominant VM cost for this workload while preserving cancellation. The paper does not extrapolate the 25.918x factor to I/O-bound handlers, unsupported functions, other processor architectures, or whole applications whose time is spent elsewhere.

7. Reproducibility identity and controls

Grayworth's performance records use an evidence-first publication rule. A headline value is admitted only after the runner has retained the command contract, executable identity, fixture identity, environment description, raw measurements, and a machine-readable completion marker. The report generator consumes verifier-accepted evidence rather than terminal excerpts or manually selected screenshots. This rule matters because a benchmark is not merely a timer around a program: it is a claim about which program ran, under what resource limits, against which request stream, and with what correctness checks. Preserving those facts makes later reanalysis possible even while Gray itself continues to change.

The HTTP experiments use an open-loop, constant-throughput load model through wrk2. In a closed-loop client, a slow response delays the next request and can make an overloaded service appear to have acceptable latency because the client stops offering work at the intended schedule. wrk2 records latency from the time a request should have been sent, exposing that coordinated-omission effect. Requested rate, achieved rate, delivery ratio, and corrected latency are therefore reported together. Throughput without latency can reward an ever-growing queue; latency without achieved delivery can describe a server that simply did not process the offered work.

Every HTTP route is preflighted for exact status, body, and content type before load begins. Warmup is separated from the accepted interval, and the server process is monitored throughout the interval. A point fails on socket errors, non-success status codes, unexpected process exit, changed fixture identity, or missing completion evidence. The one-hour matrix additionally samples resident memory and open descriptors after a settling window. Flat sampled values do not prove the absence of every leak, but they do show that the accepted workload did not produce monotonic resource growth at the recorded sampling resolution.

CPU placement is treated as part of the experiment rather than an incidental host detail. Server, load generator, controller, and database roles use declared CPU sets where the retained runner supports them. The same-host comparison constrains every server fixture to one CPU, 512 MiB of memory, and 256 processes inside one digest-pinned Ubuntu image; wrk2 runs outside that container on disjoint client CPUs. This isolates the application-facing runtime path while holding the operating-system image, cgroup contract, network path, and load generator constant.

A controlled-delivery point in the comparison is defined before interpretation as at least 95 percent of the requested rate with corrected p99 below 100 ms. It is a reporting heuristic for locating a useful throughput knee, not a universal service-level objective. The complete curve remains primary evidence because a single threshold can hide how abruptly queues form after saturation. The papers therefore distinguish the highest point satisfying that contract from the maximum achieved throughput and from the later one-hour operating point selected below the knee.

Fresh-process and interleaved sampling are used where process startup, dynamic compilation, or thermal drift could otherwise bias the result. Output is checked before a timing sample is accepted. Medians are reported for repeated compute diagnostics because they resist a single scheduling interruption, while minimum and maximum values show the observed spread. HTTP curves retain every rate point rather than reporting only the favorable region. Negative results, including the original blocking PostgreSQL tail latency and Gray's own collapse at 75,000 requested requests per second, remain in the record because they identify the architectural boundary that the next experiment must address.

The publication language separates observation from interpretation. An observed value is copied from retained output; an architectural explanation is supported by implementation structure, profiling, or an intervention; a projection is labeled as future work. Grayworth publication identifiers remain stable internal record keys, while each version 1.0 preprint also carries its reserved DOI for the archival record. A later artifact release may add source bundles without changing these canonical record URLs, DOI identities, or original measurements.

Reproducibility does not require disclosing unrelated commercial implementation details. It does require enough identity to tell whether two records describe the same executable and fixture. For that reason, the appendices retain cryptographic hashes, runtime modes, requested rates, duration, connection counts, thread counts, database versions, and named artifact directories. Public benchmark and report files can be released progressively toward Gray v1. Until then, the paper distinguishes currently downloadable manuscript and metadata files from repository artifacts named for later release. No unavailable artifact is described as independently archived or externally replicated.

Table 1. Native-loop experiment identity

Property	Retained value
Runtime SHA-256	0ba1ab4a413a2fbfa6a20abc215ca3ebba790f805d7fb0dd29fce9eeceea88a6
Fixture SHA-256	fc51a2c28ffd864afca5651c80df55a2f60d5c538aba4197421513ff3b4c4cf4
CPU	CPU 1, process-pinned
Rounds	7 interleaved per policy
VM control	GRAY_JIT=0
Expected output	4999999950000000

8. Results

All seven JIT-enabled and all seven JIT-disabled finite-loop processes produced the exact expected integer. The native samples had a minimum of 201.846 ms, median of 210.511 ms, and maximum of 259.657 ms. The VM samples had a minimum of 5,336.574 ms, median of 5,456.099 ms, and maximum of 5,618.863 ms. Even the slowest native sample remained far below the fastest VM sample, so the distributions did not overlap in the observed rounds.

The ratio of medians is 25.918. Because the benchmark includes whole-process behavior for the defined diagnostic and the native loop executes its stop-token check at every backedge, the ratio captures the implemented product path rather than a separately compiled C loop with safety mechanisms removed. The maximum native excursion increases spread but does not threaten the qualitative separation.

The cancellation diagnostic passed under Live JIT, Live VM, Static JIT, and Static VM. After the owner requested cancellation at 20 ms, the effectively unbounded loop returned through CancelledError. AddressSanitizer and UndefinedBehaviorSanitizer gates also passed. These results establish that the native backend can relinquish control through the language's structured mechanism instead of trapping the process in generated code.

The historical profile explains the scale. A VM that retires billions of instructions to execute a simple loop has substantial removable work even when its indirect branches predict well. Native lowering replaces repeated bytecode fetch, decode, dispatch, operand movement, and value handling with a compact machine loop plus one safepoint check. The observed 25.918x change is consistent with that intervention and too large to be explained by the recorded run-to-run variation.

The result does not yet match the best historical native-language arithmetic timings in Gray's broader baseline. That is expected for a first bounded backend carrying the Gray ABI and per-backedge cancellation. The decisive result is the collapse of the interpreter gap within Gray: the runtime moved one eligible region from multi-second VM execution to roughly two-tenths of a second without changing the source result or control contract.

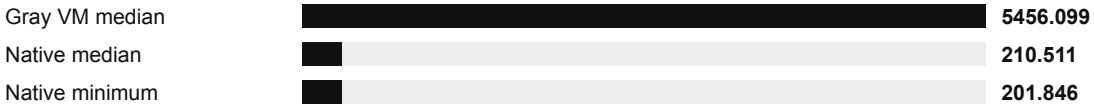
Table 2. Seven-round timing summary

Policy	Rounds	Minimum	Median	Maximum
Native enabled	7	201.846 ms	210.511 ms	259.657 ms
GRAY_JIT=0	7	5,336.574 ms	5,456.099 ms	5,618.863 ms

Table 3. Semantic matrix

Execution	Finite result	Cancellation	Safety gate
Live JIT	exact	CancelledError	pass
Live VM	exact	CancelledError	pass
Static JIT	exact	CancelledError	pass
Static VM	exact	CancelledError	pass
Native sanitizer builds	exact	CancelledError	ASan/UBSan pass

Figure 1. Normalized reported measurements



9. Analysis and interpretation

The non-overlapping timing ranges and 25.918x median ratio demonstrate that dispatch removal, not incremental tuning, changed the execution regime. The VM remains valuable as the universal semantic engine, but it is no longer required to interpret every iteration of this eligible region. The backend proves a path from Gray's integrated development model to native sustained execution without creating a second language contract.

The safety result changes the meaning of the speedup. Omitting cancellation from generated code would make the benchmark simpler and potentially faster, but it would produce a native tier unsuitable for long-running server work. By paying for a stop-token check at every backedge, Gray demonstrates that native performance and runtime governance are compatible. The cancellation test independently confirms that the check is not dead metadata.

The experiment also validates profiling-driven development. Hardware counters showed that branch misses were not the dominant issue despite the interpreter's high branch count. That evidence rejected an alluring but insufficient focus on prediction tricks. Native lowering attacks total dispatch and instruction volume. The large measured response supports the original diagnosis and establishes a basis for expanding eligibility.

The comparison is internal by design. Cross-language timings can reveal strategic gaps, but they combine different compilers, integer semantics, startup models, and benchmark implementations. Gray native versus Gray VM keeps source, output, runtime build, process policy, and machine constant. It isolates the value of the new execution tier, which is the causal question this paper can answer most strongly.

10. Engineering implications

Gray can preserve its interpreter as a correctness and coverage foundation while moving proven hot shapes into bounded native regions. This reduces pressure to encode increasingly specialized fused bytecodes and gives the compiler a place to exploit typed information directly. Eligibility can expand by value class, operation, call form, and control shape, each guarded by differential tests against the VM.

Server behavior must remain part of that expansion. Every loop and call path needs an explicit cancellation and ownership story. Safepoints at backedges provide a simple initial rule that is easy to audit. Future work may add function-call polls, allocation polls, or bounded polling intervals, but changes should be measured against both throughput and cancellation latency rather than treating checks as overhead to erase.

Static embedding turns native lowering into a deployment feature rather than a warm-process trick. Eligible machine code can travel with the artifact and be relocated at startup, while Live mode can continue to favor iteration. The common ABI and semantic matrix make performance policy selectable without making application behavior mode-dependent.

The result also reframes Gray's broader benchmark history. Earlier comparisons showed competitive startup and server behavior but a clear sustained-compute gap. The native loop closes a large portion of one internal gap in a single architectural step. It does not finish a general backend, but it demonstrates that the bottleneck was tractable and that Gray can make dramatic progress without surrendering the integrated runtime properties that distinguish it.

11. Threats to validity

The principal limit is workload coverage. One arithmetic loop is not a general application suite. Allocation, strings, calls, branches with complex control, foreign functions, database operations, and network handlers may spend little or no time in eligible native regions. The 25.918x ratio applies to the reported function and policies. It should be used as evidence that the mechanism works, not as a multiplier for unrelated programs.

Seven rounds provide a clear separation but not a detailed statistical model of host variance. CPU pinning, fresh processes, interleaving, and exact output checks control obvious biases. More rounds, hardware performance counters on the final binary, and repetition across processors would improve precision. The reported minimum and maximum expose the observed spread rather than hiding it behind the median.

The historical perf profile and current timing run use different retained stages, so instruction counts are explanatory rather than a before/after counter comparison. A future matched study should collect retired instructions, branches, cache behavior, and cycles for both current policies. The causal timing result remains strong because the policy switch operates on the same current function and runtime identity.

x86-64 is the only backend covered here. ABI behavior, instruction selection, memory model, and safepoint implementation require separate validation on other architectures. Sanitizers improve confidence in the exercised paths but do not prove all generated instruction sequences. Finally, commercial source is not publicly included in version 1.0; exact hashes preserve identity until the planned Gray v1 artifact bundle enables broader replication.

12. Artifact availability

The permanent GW-2026-0004 record provides this manuscript, PDF, citation exports, metadata preview, and related newsroom link. Retained repository evidence includes benchmarks/jit-loop-report.md, the finite jit-loop.gy fixture, the cancellation fixture tests/jit-loop-cancel.gy, and the historical sustained benchmark and perf records described in BENCHMARKS.md.

The Gray v1 artifact release is intended to publish releasable fixtures, exact runner commands, raw round timings, sanitizer commands, and build identity material. The version 1.0 preprint uses DOI 10.5281/zenodo.21633227; future code and benchmark attach-

ments will preserve this paper's Grayworth ID, DOI, canonical URL, author, manuscript date, and numerical record.

13. Conclusion

Bounded native loop lowering reduced the median whole-process time of Gray's eligible 100-million-iteration sum from 5,456.099 ms in the VM to 210.511 ms, a 25.918x speedup. All accepted runs produced the exact same 4,999,999,950,000,000 result. The observed native and VM ranges did not overlap across seven interleaved rounds.

The backend achieved that result while executing a cooperative stop-token safepoint at every generated backedge. A separate effectively unbounded loop responded to cancellation with CancelledError across Live and Static, native and VM policies, with sanitizer gates passing. Gray therefore did more than make one loop fast: it demonstrated that a young integrated runtime can remove its dominant interpreter cost while retaining the control semantics required by a production server system.

14. References

1. Gil Tene. wrk2: a constant throughput, correct latency recording variant of wrk. https://github.com/giltene/wrk2
2. PostgreSQL Global Development Group. Asynchronous Command Processing in libpq. https://www.postgresql.org/docs/current/libpq-async.html
3. The Go Authors. Package net/http. https://pkg.go.dev/net/http
4. The Go Authors. Diagnostics. https://go.dev/doc/diagnostics
5. Node.js Project. Don't Block the Event Loop or the Worker Pool. https://nodejs.org/en/learn/asynchronous-work/dont-block-the-event-loop
6. PHP Documentation Group. FastCGI Process Manager configuration. https://www.php.net/manual/en/install.fpm.configuration.php
7. M. Anton Ertl and David Gregg. The Structure and Performance of Efficient Interpreters. https://jilp.org/vol5/v5paper12.pdf
8. LLVM Project. Garbage Collection Safepoints in LLVM. https://llvm.org/docs/Statepoints.html
9. Revised6 Report on the Algorithmic Language Scheme. https://r6rs.org/final/html/r6rs/r6rs.html
10. Grayworth. Gray comparison methodology and retained dataset. https://grayworth.com/compare/methodology
11. Grayworth. Gray performance research record. https://grayworth.com/research
12. Grayworth. Formal HTTP comparison data. https://grayworth.com/data/http-comparison-results.json

Appendices

Appendix A

Appendix A records immutable identity. Runtime SHA-256: 0ba1ab4a413a2fbfa6a20abc215ca3ebba790f805d7fb0dd29fce9eeceea88a6. Fixture SHA-256: fc51a2c28ffd864afca5651c80df55a2f60d5c538aba4197421513ff3b4c4cf4. The finite expected result is 4999999950000000. The cancellation bound is 9000000000000000000 with cancellation requested after 20 ms.

Appendix B

Appendix B defines the reported ratio. Native median is 210.511 ms. VM median is 5,456.099 ms. Dividing VM by native yields 25.918 after rounding. The ratio is not formed from the fastest native and slowest VM samples, and no samples are removed from the seven accepted rounds after output validation.

Appendix C

Appendix C records semantic scope. The native backend preserves the Gray runtime ABI, source-function identity, structured exits, exact integer behavior for the fixture, and cancellation return. A stop-token poll executes at every generated loop backedge. Unsupported functions remain in VM execution. No claim is made that the backend currently lowers arbitrary Gray programs.

Appendix D

Appendix D places the result in the compiler roadmap. The measured backend is a baseline native tier, not a claim of completed global optimization. Its importance is that it establishes the hard interfaces early: eligibility, code ownership, ABI entry, value transfer, source identity, structured return, and cooperative interruption. Additional arithmetic forms, branches, calls, allocations, and architectures can be admitted behind those interfaces and compared differentially with the VM. This order keeps semantic coverage independent from optimization coverage. A function that does not satisfy the native contract remains correct in the interpreter, while a function that does satisfy it can remove dispatch from its hottest repeated region. The 25.918x result supplies quantitative evidence that expanding this tier is worthwhile.