RTBLint: Executable, Versioned Contracts for OpenRTB Interoperability

Aleksander Sekowski

Independent Researcher
rtblint.org · github.com/aleksUIX/rtblint

Revised author-review preprint, November 2026. Frozen release: RTBLint 0.14.7. A separately identified research candidate and admission policy are evaluated alongside it.

Abstract

OpenRTB interoperability depends on a specification snapshot, a wire encoding, an integration direction and sometimes an originating request. We present RTBLint, a Rust system that selects executable contracts across these dimensions, and an evidence-preserving admission wrapper that separates rejection, unsupported scope and passage of selected checks. Profiles can authorize documented exceptions as well as impose requirements, so composition cannot be modeled as an ever stricter sequence of schemas.

The evaluation distinguishes development evidence, author-derived protocol tests and externally authored receiver fixtures. On 77 original cases within common scope, a separately implemented, context-specialized schema checker agrees with all stated expectations. Frozen RTBLint agrees with 75; its two disagreements concern malformed Native JSON reported as warnings. Thus the advantage over a context-free schema is attributable to supplied context and implemented obligations, not an intrinsic inability of schemas to express them. A seeded 792-case challenge exposes eight additional decisions affected by two omitted protobuf root flags. A two-field research patch repairs those failures. With an explicit Native-warning policy, the candidate gate agrees with all 80 original binary labels and all 792 reviewed challenge labels, while marking five original scope controls indeterminate. These directed cases do not estimate production accuracy.

An external challenge imports all 140 fixtures from four predeclared Prebid Server suites, with maintainer-authored HTTP expectations. The static release rejects 57 of 106 expected rejections and accepts 27 of 34 expected acceptances. The disagreements expose normalization, configuration and unimplemented receiver rules, rather than supporting a universal compatibility claim. A production-handler replay matches all 140 HTTP expectations. For the 34 accepted transactions, request passage under the same Prebid contract rises from 27 before normalization to 32 afterward. Native/WASM parity on 3,846 byte-identical release cases and 18.28 million additional timed calls provide implementation and cost evidence. Eleven shuffled blocks in macOS and Linux environments on the same M4 machine place one-request median block p95 at 2.13 and 1.92 microseconds respectively. The artifact includes the defect, repair, comparison checker, complete disagreements and a usable admission ledger.

Keywords: OpenRTB; protocol validation; JSON Schema; interoperability; conformance testing; WebAssembly; executable contracts.

1. Introduction

A bid response can be valid JSON, satisfy every local field constraint and still answer the wrong auction. Its `id` may belong to another request. Its `impid` may refer to an impression that was never offered. Its currency may be outside the request's allowed set. The information needed to identify these inconsistencies is absent from the response itself. No improvement in the speed of a response-only validator can recover evidence it was not supplied.

A different problem occurs before any cross-message check. OpenRTB integrations share field names without necessarily sharing their complete wire contracts. The canonical specification uses integer flags in many places. The IAB protobuf model declares a set of corresponding fields as booleans, which acquire a different JSON representation. Google's documented OpenRTB JSON integration adds its own serialization and extension conventions. Prebid Server's publisher-facing auction endpoint adds bidder-selection and stored-request conventions. A field that is acceptable at one boundary can therefore be a useful diagnostic at another [1, 2, 3, 4].

These differences make the question "is this OpenRTB valid?" underspecified. A useful answer must identify which revision, representation and integration boundary was checked, and what evidence was available. It must also distinguish a missing implementation capability from a demonstrated error in a message. Otherwise a successful check against a narrow contract can be mistaken for approval of an entire integration.

RTBLint is an open-source system for checking these declared contracts [18]. It combines catalogs extracted from specification field tables with executable semantic rules, a wire-dialect selector, integration profiles and an optional originating request. The system returns structured diagnostics containing a stable rule identifier, severity, affected path and an optional specification section or source URL. The same core is available natively and through WebAssembly. This paper studies the design and the behavior of a frozen build, rather than the prevalence of errors in deployed advertising traffic.

Two examples motivate the architecture. In a Google Authorized Buyers request, an auction type authorized by Google's documentation can be rejected by a canonical enum check. Selecting the appropriate profile should remove that mismatch, while retaining relevant checks on Google's required extensions. In a paired response, all locally required fields can be present while the referenced impression is absent from the request. Supplying the request should add a finding. A correct contextual validator must therefore both retract inappropriate canonical findings and introduce findings that a local check cannot justify.

The paper asks four research questions:

1. **RQ1, contract correctness.** Which source-derived obligations are implemented correctly, which tests expose defects, and does a small repair remove the demonstrated failures?

2. **RQ2, context and mechanism.** Does the detection advantage over a local schema persist when a separate schema-based checker receives equivalent context?

3. **RQ3, receiver boundaries.** How do static decisions relate to independently authored receiver expectations and actual behavior of a production endpoint with local adapters?

4. **RQ4, implementation cost.** Are release interfaces consistent on identical bytes, and what variation appears when native timing order and execution environment change?

The contributions are an explicit contract and evidence model, an executed comparison that separates information from validation mechanism, a falsifiable repair for a discovered dialect defect, and a practical admission artifact exercised against a production server handler. The external challenge deliberately includes every fixture in four pinned suites. Its labels remain receiver expectations, rather than being recast as universal protocol truth. The systems contribution is the organization, testing and operationalization of contextual checks. Neither OpenRTB validation nor paired-message checking is claimed as a new idea.

The central scope is OpenRTB 2.x JSON with selected profiles. The system also includes layered OpenRTB 3.0 validation, but this paper does not establish comprehensive 3.0 behavior. The receiver experiment uses one pinned implementation and its local test infrastructure. It does not sample a population of exchanges, production traffic or advertiser outcomes. Earlier specification-checkability and live-traffic studies address different empirical questions [5, 6].

2. Background and related work

2.1 Protocol contracts exist at several boundaries

OpenRTB defines a request describing offered impressions and a response carrying bids. Its field tables describe types, required members, nested objects and enumerated values. Implementation prose adds relationships and operational guidance. Beginning with dated 2.6 revisions, a major.minor header does not identify every additive snapshot. A reproducible validation run must therefore pin its specification revision separately from the header [1].

The message also has a role. A publisher-facing auction request and a bidder-facing request occupy different boundaries, even when both use OpenRTB objects. Prebid Server, for example, documents enrichment, stored requests and publisher-facing extensions. Some references in an observed message may be supplied or expanded elsewhere. Absence from a captured request is consequently different from proof that the relevant value does not exist anywhere in the integration [4].

This distinction matters for error interpretation. An unknown field outside `ext` can indicate divergence from a producer catalog without demonstrating that a tolerant receiver will reject the message. A deal identifier absent from a captured request can require out-of-band verification. Neither should silently become a claim about transaction failure. The system's diagnostic policy and the experiment's interpretation must identify which statement is being made.

2.2 JSON Schema supplies a substantial local baseline

JSON Schema supports object structure, required members, numeric and string constraints, enums, dependencies, conditionals and composition. It is substantially more expressive than a simple type checker. Draft 2020-12 also defines vocabulary behavior for string-encoded content; the presence of content annotations alone does not imply that an implementation validates arbitrary embedded protocols [7, 8].

The comparison in this paper is against specific schemas and their selected entry points. It is not a proof that semantic or version-specific constraints cannot be written in JSON Schema. Some could be added to the baselines. A schema can also be generated for a particular revision or profile. The engineering issue is which constraints are represented, what context is supplied and how unsupported scope is communicated.

Work on modern JSON Schema formalizes its evaluation model and complexity [9]. Blaze studies compilation and optimization of JSON Schema validation [10]. These results make it particularly inappropriate to infer that a domain-specific validator is inherently faster than schema validation. Our schema experiments compare decisions. Our performance experiments characterize RTBLint's native API. They do not compare a Rust implementation's speed with a Python schema engine and call the result an algorithmic advantage.

2.3 OpenRTB validation predates RTBLint

The `openrtb/openrtb2x` project includes an OpenRTB validator and describes support for earlier protocol versions. A separate open-source project by Klaus Silveira provides an OpenRTB 2.6 JSON Schema with request and response definitions [11, 12]. These are relevant predecessors. This paper makes no claim to introduce the first machine-readable OpenRTB contract or the first OpenRTB validation API.

The independent schema is useful experimentally because it does not come from RTBLint's catalog pipeline. RTBLint's own generated schemas provide a different control: they share the system's field metadata but omit parts of its executable logic. Agreement with the internal baseline cannot establish that the shared source interpretation is correct. Comparing both baselines helps separate shared catalog structure from additional contextual behavior, although neither baseline is an external ground-truth oracle.

2.4 Encoding and profile are separate parameters

The protobuf JSON mapping represents booleans differently from integer flags and gives special treatment to wide integers, names and other types. The IAB OpenRTB protobuf declarations provide the corresponding field types [2, 3]. RTBLint exposes a selected encoding rather than guessing it from isolated values. Its published divergence table contains 28 object/field pairs.

Selecting that dialect is not the same as declaring a vendor profile. Google's OpenRTB JSON conventions, for example, must be read from Google's guide; they do not follow automatically from seeing a protobuf reference beside a field. Google's guide specifies decimal-string JSON encoding for 64-bit integer fields [13]. The distinction appeared in our own fixture construction: three intended positive controls initially used a numeric billing identifier where the documented representation was a string. Section 5 retains the correction history.

The dialect feature evaluated here addresses specific type and name differences. It should not be read as a claim that RTBLint implements every acceptance rule of every ProtoJSON parser. The study does not exhaust null handling, duplicate keys, every enum serialization option or arbitrary generated protobuf messages.

2.5 Earlier empirical work and the missing systems question

Our earlier checkability study classifies specification statements by the evidence and machinery needed to check them [5]. Our live-traffic study measures client-side header-bidding dialects and includes paired and creative-layer analyses [6]. Their questions concern the specification's enforceability and observed traffic. The present paper asks how a particular implementation selects, combines and exposes executable contracts, and what evidence supports its behavior and cost.

Chaudhary's VAST study derives player duties from a specification and grades client behavior from recorded requests [14]. It cites our earlier VAST validation work [19] and tests a VASTLint client runtime. Its main methodological relevance here is the separation between document validity and runtime behavior. A payload validator does not observe whether a client later sends the correct tracking request or whether an auction server respects a deadline.

The HTTP Garden similarly makes transformations and interpretations visible at different protocol boundaries [15]. The protocol-maintenance argument in RFC 9413 stresses the value of identifying and resolving ambiguity rather than allowing tolerances to become permanent, undocumented conventions [16]. These perspectives motivate explicit contracts and replayable evidence. Our main experiments concern message-level contracts. The receiver experiment adds a bounded check of one production handler under controlled local conditions. A population study of independent auction servers remains separate work.

3. A model of executable validation contracts

3.1 Context and evidence

We describe a validation context as a tuple:

```
C = (v, d, p, t)
```

Here `v` is a dated specification snapshot, `d` a declared JSON dialect, `p` an integration profile including its direction, and `t` the payload role. The input is the original message string `x`, not merely a parsed JSON object. Optional evidence `E` can include the originating request or locally supplied reference artifacts.

The system computes a finite collection of diagnostics:

```
F(C, x, E) = { (rule_id, severity, path, section_or_source) }
```

Human-readable messages accompany these fields in the implementation. This notation describes observable behavior; it does not assert that the implementation is a verified proof system. The public boolean is derived from severity:

`valid(C, x, E) = true` exactly when `F(C, x, E)` contains no error-severity finding.

The definition explains both an advantage and a limitation. Warnings can convey useful evidence without forcing rejection. But the boolean cannot, by itself, distinguish a clean, fully applicable check from a warning about unsupported scope. Consumers need the context, diagnostics and applicable coverage boundary.

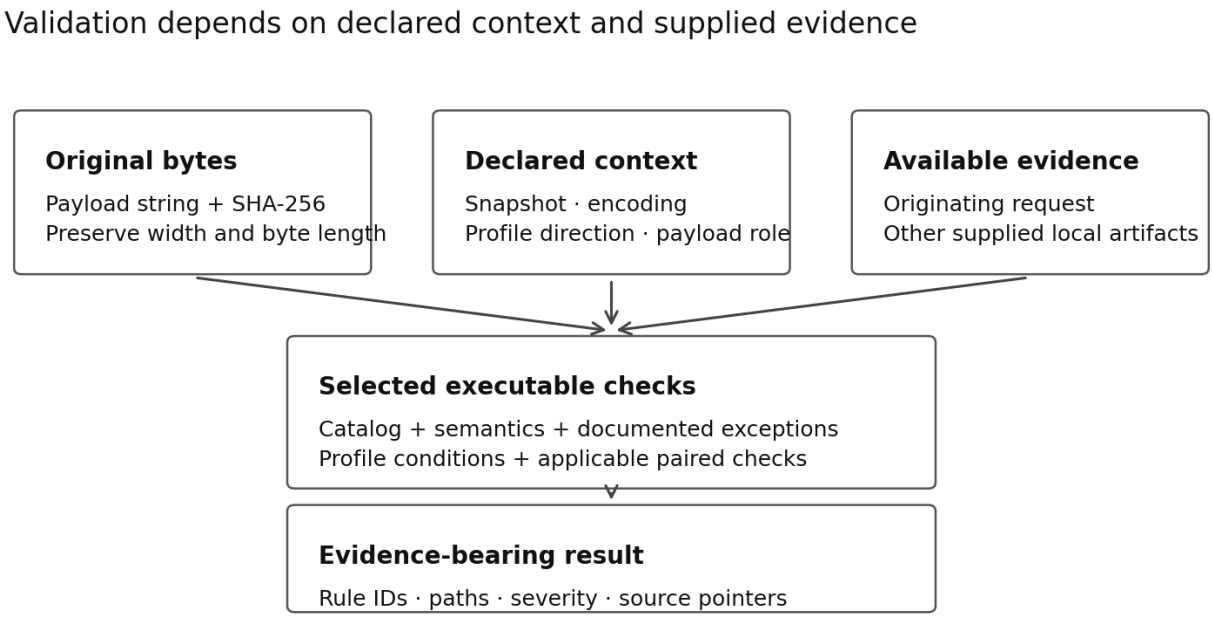


Figure 1. The contract model separates original message bytes, declared context and optional evidence. Source-aware selection includes both additional checks and documented compatibility exceptions. This is a conceptual model of the implemented responsibilities, not an exact call-order trace.

3.2 Profiles are contextual overrides as well as additions

Let $A(C)$ denote the messages receiving no error-severity finding under a context. A simple model in which each added layer only restricts the accepted set would require every profile's accepted set to be a subset of the canonical set. That model does not describe the implemented behavior.

A documented Google fixed-price request is a witness. The canonical catalog rejects its auction-type value; the selected Google profile authorizes it. Conversely, an otherwise structurally acceptable request can fail a profile because a documented integration field or bidder selection is missing. The profile changes the applicable contract, rather than merely increasing strictness.

The implementation therefore contains both additions and compatibility exceptions. Canonical enum checks consult profile-authorized values. Required-field and shape checks can consult documented exceptions. Profile-specific checks inspect extensions and relationships that the generic catalog leaves open. The paper uses "composition" for this selection and combination behavior, not for an externally programmable rule language.

The choice has an important consequence for evaluation: fewer findings are not automatically better, and more findings are not automatically more correct. Every change needs a source-backed expectation under a declared context. A profile can correct an inappropriate rejection while introducing a justified rejection elsewhere.

3.3 Evidence limits must survive composition

Some obligations are directly decidable from the supplied message. Others need a second message or artifact. If the originating request is missing, the response's `impid` cannot be checked for membership in that request. If a stored request has not been expanded in the capture, its missing material may prevent a pairing conclusion. If only a VAST wrapper is present, eventual creative properties may remain unavailable.

We distinguish three conditions conceptually: a check produced evidence of a mismatch; a check found no mismatch in available evidence; or the evidence or implemented scope was insufficient. RTBLint's current API does not encode these as a universal three-state result type. It uses errors, warnings and explicit unsupported diagnostics in selected paths. The distinction must therefore be retained in a surrounding execution record rather than inferred from the boolean alone.

This is an evidence boundary, not a claim that all unavailable obligations are undecidable in principle. The limitation may disappear when another message, an expanded request or a runtime observation becomes available. A diagnostic should state what cannot be established from the current evidence instead of silently treating absence as success.

3.4 Original representation belongs in the contract

Most field constraints concern the decoded JSON value. Other checks concern the original representation. A published byte limit applies to the bytes sent on the wire. Reformatting the document can change the relevant quantity. Parsing large integers into JavaScript numbers can change their values before the validator sees them.

The reproducibility record therefore stores the raw message string and a SHA-256 digest, along with the selected context. Parsed objects remain useful metadata for inspection and schema validation, but they are not a replacement for the input. Paired runs retain both strings. This requirement is experimentally consequential: two harness corrections described in Section 5 were necessary before a native/WASM comparison could meaningfully test the implementation.

3.5 A bounded evidence contract

A contract should state which conclusions are justified, not only which parameters select the code. Let an obligation be a tuple `o = (source, applicability, required_evidence, predicate)`. Given requested obligations `Q`, implemented obligations `S(C)` and observations `E`, the assessable subset is `T = {o in Q intersect S(C): applicability(C, x, E) and required_evidence(o) is available in E}`. Let `N` contain requested obligations whose applicability is demonstrably false. The unassessed set is `U = Q minus (T union N)`. Obligations in `N` are inapplicable, rather than failed or unassessed. A finding is evidence about an obligation in `T`; absence of a finding establishes no claim about obligations in `U`.

This formulation gives a simple limit on composition. If two executions receive identical evidence concerning an obligation that needs an absent originating request, no deterministic rearrangement of local checks can distinguish responses that differ only in whether their impression reference existed in that missing request. The missing evidence, rather than the choice of schema or procedural code, is the obstacle. Once the request is supplied, membership can be expressed procedurally or by specializing a schema with its permitted values. RQ2 tests precisely this alternative implementation.

A second limit concerns nonmonotone profiles. Adding evidence to a fixed contract may enable further checks. Selecting a different contract can also retract a canonical diagnostic, as the Google auction-type example shows. We therefore do not impose a monotonicity law across different profiles. Provenance identifies both the selected contract and the evidence that made each conclusion available.

For an executable deployment policy, we use three dispositions: `reject` if a supported check demonstrates a blocking violation, `indeterminate` if required context or declared scope is unavailable without such a violation, and `pass_selected_checks` otherwise. The last name is intentional: it does not claim that every obligation in a specification was assessed. This policy provides decision discipline, not a proof that the underlying predicates are sound. Incorrect predicates and incomplete capabilities remain empirical failure modes, including the defect found in Section 6.

4. Implementation

4.1 Frozen system and catalog representation

We evaluate RTBLint 0.14.7 at commit `4defcb383729e20147a82ca5737974be03426192`. The working tree was clean when archived. The source archive and evaluation driver are included with the artifact. No validator source was changed during the study.

The compiled inventory tracks 18 OpenRTB snapshots. Seventeen have usable validation catalogs; the April 2022 2.6 snapshot has no usable object catalog and receives an explicit unsupported-version diagnostic. The latest evaluated 2.6 catalog contains 38 objects and 417 field definitions. The system exposes 27 profile identifiers, including the canonical specification profile. The other 26 describe vendor or producer directions and must not be counted as 26 distinct exchange companies.

Catalog interchange files store object names, field names, type annotations, child-object references, value sets and citations. A build step turns them into static Rust structures. Shapes, required and deprecated flags, and child-object relationships are available directly at runtime; the validator does not reparse specification prose for each message. Versioned path data and AdCOM list values supply additional checks.

Generated JSON Schemas and the native walk share this metadata. This reduces one kind of drift while preserving one kind of common-mode error: a wrong catalog entry can affect both. Build-time generation is therefore an implementation strategy, not independent evidence of source correctness.

4.2 Validation stages

Before the normal 2.x walk, the implementation checks whether the selected profile supports the payload direction and version family. An unsupported profile direction falls back to canonical validation and appends `openrtb.profile.scope.unsupported`. A snapshot without a usable root catalog returns `openrtb.version.unsupported`.

For an applicable 2.x context, the system parses the original input with `serde_json`, verifies the root shape, applies any raw-response checks and traverses known objects. The walk checks required fields and arrays, declared shapes, enumerated values and versioned path status. It applies local

semantic rules and selected profile rules at relevant objects. Unknown `ext` content remains open unless a selected profile supplies a more specific contract.

The profile hooks participate at several points. They are not one independent pass whose result can always be appended to an unchanged canonical report. A documented exception can affect whether a canonical field, enum or shape diagnostic should exist. Object-specific rules then contribute constraints that the catalog cannot express by itself.

The walk maintains an instance-path cursor and constructs paths when findings are emitted. Its performance depends on both input parsing and the number and type of visited objects. A large opaque string and a similarly sized array of structured impressions are consequently different workloads.

4.3 Paired response validation

The paired API first performs full standalone response validation. It then parses the originating request, indexes relevant impression and constraint context, and checks response references and relationships. These include request identity, impression references, offered media, seat and currency restrictions, and selected Native asset relationships. Some deal-related findings explicitly request verification of out-of-band arrangements rather than treating every missing reference as a definite protocol violation.

The request supplied as context is not itself fully validated by the paired API. A caller seeking a complete request report must validate it separately. The distinction avoids confusing "used as reference context" with "accepted by the request contract". The evaluation records those roles explicitly.

Paired validation is implemented for OpenRTB 2.x. For OpenRTB 3.0, the API performs standalone response validation and emits a warning that paired scope is unsupported. The existence of layered 3.0 transport and domain checks does not imply the existence of the same cross-message checks in that family.

4.4 Nested protocols and diagnostic policy

Native request and response content may arrive as JSON encoded inside an OpenRTB string. The core can inspect this nested representation and, when a request is supplied, check selected required-asset relationships [17]. Markup classification also supports response/request media-coherence checks. These capabilities do not establish that arbitrary creative content is renderable or that remote resources will succeed.

Diagnostic identifiers are stable machine-facing keys. Severity determines the exported boolean. `section` can identify a specification section or a profile source URL, but it is optional and is not a universally complete provenance object. It does not itself include the selected version, source digest, dialect or integration direction. The artifact's execution manifest supplies those fields alongside the result.

This separation is visible in the Native parsing result: unparseable embedded JSON is reported, but as a warning. The result is informative to an observer and insufficient for a consumer that equates

`valid: true` with satisfaction of every nested contract. We retain that behavior as measured, rather than adjusting the implementation to improve a headline score.

4.5 Interfaces and cost boundaries

The evaluated native driver calls the public Rust API directly. The distributed Node package calls a WebAssembly build and returns the same diagnostic structure. The distribution comparison uses the package committed in the frozen repository, which reports core version 0.14.7; its binary digest is recorded independently of the source commit.

The latency experiment measures the native API, including parsing, check execution, result construction and disposal. It excludes process startup, file I/O, network requests and JSON serialization of the returned report. It is therefore relevant to an embedded call in a warm process, not a command spawned once per bid. We make no comparative performance claim for WASM, a remote service or a full auction pipeline.

4.6 Evidence ledger and research candidate

The accompanying `admit.py` implements the bounded admission policy for declared scope, required pairing and a named severity policy around the native research driver. It does not enumerate the entire specification obligation set or discover every member of `U`. Its passage label therefore remains limited to selected executed checks. It records version, dialect, profile, role, pairing status, policy identity, driver digest and the SHA-256 digest of each original UTF-8 input. The CLI reads bytes before decoding, preserving CRLF line endings for hashing. A paired invocation validates the originating request separately as well as checking the response against it. An invalid originating request is therefore not silently used as trustworthy context.

The `core-errors` policy follows blocking core findings. The `strict-nested` policy additionally promotes only `openrtb.native.request.unparseable` to a blocking condition. Unsupported version or profile-direction diagnostics produce `indeterminate` when no supported violation already requires rejection. The wrapper also supports an explicit requirement for paired evidence. Missing required pairing is indeterminate, while a demonstrated violation still takes precedence. Exit codes are 0 for passage, 2 for rejection, 3 for indeterminate and 4 for execution failure. An execution failure never becomes passage.

This wrapper is research software included in the artifact, rather than a claim about an already released API. The frozen product retains its original boolean semantics. A separate candidate adds `BidRequest.allimps` and `BidRequest.test` to its existing sorted protobuf-boolean table. Its base commit, complete patch and patch digest are recorded. The public divergence count becomes 30 instead of 28; its inventory test is updated to assert both newly included fields. Four other booleans extracted from the protobuf source belong to its nested Native model and are not counted as covered OpenRTB envelope fields. The candidate is not described as a complete ProtoJSON implementation.

5. Evaluation method

5.1 Frozen study, provenance and units

The study protocol is stored in `evaluation/study-protocol.json`, with digest `2e8b5c7fa4fbfef3791216bfcfb17c57efe16f243ef99c32b1ae6a472a07b9e9`. It was recorded before execution of the added challenges. It declares complete-suite inclusion, equivalent-context comparison, seeded generation, a local production-handler experiment and 11 shuffled timing blocks. It is a local frozen protocol, not an externally registered preregistration. Correction records identify subsequent source-driven repairs to the harness and generated inputs.

The release remains RTBLint 0.14.7 at commit `4defcb383729e20147a82ca5737974be03426192`. The research candidate consists of that source plus the archived two-field patch and the corresponding inventory-test update. The original product checkout is not modified. We keep three kinds of evidence separate because their expected outcomes have different origins.

Table 1. Evidence sets and the interpretation of their labels.

Set	Units	Origin of expected outcome	Permitted inference
Original contrast set	85 rows, 80 binary labels, 5 scope controls	Study author, source-derived	Targeted contract behavior
Seeded challenge	792 rows from 128 paired contexts and a flag grid	Study author, declared obligations	Bounded combinatorial behavior and defect exposure
External receiver challenge	140 complete upstream fixtures	Prebid maintainers' HTTP expectations	Compatibility with that receiver's fixture contract
Committed development replay	3,761 rows, 3,579 binary labels	Frozen system repository	Regression consistency
Metamorphic executions	308 transformed rows	Relation to original decision	Observed invariance under named transformations
Timing study	84 configurations in 11 blocks per environment	No correctness labels	Local processing cost and observed variation

A row is a test execution, not an independent sample from deployed traffic. The original 85 rows collapse to 67 templates after identifier normalization. The 768 paired challenge rows reuse 128 request contexts. Millions of timed calls likewise do not constitute millions of independent hardware observations. We report the relevant denominator rather than pooling these sets into a headline accuracy number.

5.2 Equivalent-context reference implementation

The first edition compared local schema checks with increasingly contextual RTBLint modes. That identifies what added information and rules detect, but confounds information availability with validation mechanism. The revised primary comparison therefore supplies both implementations with the version, dialect, profile, payload role, original payload string and originating request when present.

The reference checker imports the independently authored OpenRTB schema at commit `b81b8acf71349ed60333b370517bf417fdace12b` [12]. It imports no RTBLint catalog, generated schema, rule implementation or diagnostic table. Python `jsonschema` 4.25.1 evaluates its Draft 7 schemas. A direction-bound root reference avoids treating a request that satisfies a response schema as a successful response check. The supported comparison scope is OpenRTB 2.6-202606 with `spec`, `google-ab` or `prebid-server`, not every RTBLint profile.

The checker separately specializes that schema from the supplied context. Request identity, impression membership, allowed currency, seat selectors and offered media become constants, enumerations or conditional schemas derived from the originating request. Integer/boolean flag representations are derived from the pinned IAB protobuf source. Source-derived additions cover inventory exclusivity, impression media presence, discrete-duration bounds, selected Google and Prebid obligations, and nested Native decoding and required assets. Obligations and primary source URLs are archived in `reference-obligations.json`. Supplementary Python predicates handle embedded JSON and selected profile conditions. Consequently this is a schema-centered executable checker, not a claim that a single untouched static schema expresses every rule.

This implementation is separate, but its study author is also the system maintainer. It is not a blinded human oracle. Preliminary execution exposed an absent `mtype` enumeration in the external schema; the reference adds the four specification values. A flag-grid run exposed its absent canonical 0/1 bounds; those are added from the same pinned primary source. Both corrections are retained. No case was relabeled to agree with the checker. The strongest independence claim concerns the external schema and upstream fixture authorship, not the author's interpretation of every added predicate.

We evaluate 77 original binary cases within common scope and all 792 reviewed generated cases. A checker outside its declared scope returns unsupported rather than passage. We retain local-schema and RTBLint layer results as explanatory ablations. They do not establish that procedural validation is more expressive or inherently more correct than contextual schema specialization.

5.3 Seeded challenge, invariants and fault sensitivity

Seed 20261110 constructs 32 request contexts at each of 1, 2, 4 and 8 impressions. Each context supplies one valid response and five negative responses that independently alter auction identity, impression reference, currency, seat or offered media. This produces 768 paired rows. A further 24 rows cross the two root flags `test` and `allimps`, two declared dialects and six wire values: 0, 1, 2, false, true and the string "1". Expectations follow the specification's flag meanings and the pinned protobuf bool declarations [1, 2]. These are bounded, author-derived cases, rather than sampled production messages or exhaustive protocol verification.

The initial generator placed both `wseat` and `bseat` in the same originating request. Separate request validation in the admission wrapper exposed that specification violation. The reviewed manifest alternates a single allow-list or a single block-list by seed index, preserving all response labels. Both manifests, the first results and the correction rationale remain archived. The final 792-row result is therefore an author-reviewed challenge, not an untouched held-out result. This distinction prevents a correction made after feedback from being presented as independent predictive success.

For metamorphic checks, we transform applicable original cases by changing key order, whitespace, consistently renaming envelope identities, or reversing impression order. The expected relation is invariance of the boolean under the same declared contract. We do not assert diagnostic-order or path invariance. Malformed JSON that cannot support a transformation is excluded with an explicit parsing condition. The resulting 308 rows yield 1,232 comparisons across four modes.

We also disable each of five paired-obligation families in the reference checker. A challenge kills a deliberately injected fault if any unchanged case distinguishes the faulty checker from its full version. This is a narrow assessment of challenge sensitivity. It is not mutation coverage of the Rust implementation, proof of oracle correctness or an accuracy estimate.

5.4 Independently authored receiver challenge

The external challenge is pinned to Prebid Server commit `46670d63b5b749cb4d52eb0d2b0a801ee28c7721` [20]. Inclusion is every JSON fixture recursively within `valid-whole/exemplary`, `valid-native`, `invalid-whole` and `invalid-native`, excluding filesystem metadata. The 140 fixtures and their maintainer-authored `expectedReturnCode` fields are frozen before RTBLint execution. The manifest digest is `b72d2a7248276c42568856a3462a0fe2b4d542ff10fd0a7ef0fa0b0b1a03ad90`. Expected HTTP codes, rather than directory names, determine the labels: 34 expect 200 and 106 expect 400.

We select the declared OpenRTB 2.6-202606 / spec-json / Prebid request contract for all static comparisons. Some upstream fixtures explicitly exercise conversion between versions. Consequently a disagreement may identify the capture boundary or normalization contract, rather than a universal error in either implementation. The fixture commit is later than the earlier Prebid source boundary used by the product profile, commit `fa73b89049767eaab6d9f48dc3b0a275db563233`. Third-party authorship does not establish that the fixtures were unseen during product development. We preserve that temporal difference and do not silently update the release profile to fit the challenge.

A research `_test.go` file runs the actual auction HTTP handler and exchange code using upstream `parseTestData` and `buildTestEndpoint`. It preserves fixture configuration, account requirements, blocked-app rules, mock bidder setup and stored-request test infrastructure. Requests enter through

`httptest.NewRequest`; the handler runs in process. Currency and bidder test servers communicate over loopback HTTP. Generic mock adapters replace commercial bidder adapters, so this does not evaluate real bidder serializers, remote bidder acceptance or production configuration.

For every fixture, the recorder captures the exact incoming request string and digest, actual HTTP status and response body, the normalized request reaching `HoldAuction`, and available outgoing mock-adapter requests. Assertions compare the actual status with the upstream expectation and check an expected error substring when supplied. The static corpus initially stores a compact encoding of the fixture object. The integration comparison separately uses the handler's captured original string, preserving its actual bytes. Complete differences remain inspectable rather than being discarded as outliers.

5.5 Development replay and representation checks

We retain the first edition's replay of 3,761 committed cases from 15 manifests. Of these, 3,579 have binary expectations; the others specify required or forbidden diagnostics or scope conditions. This is useful regression evidence but weaker evidence of independent correctness. The original native/WASM comparison also remains: all 3,846 original and replay cases receive identical raw strings, context and paired strings, with SHA-256 verification before comparing complete finding multisets.

The first edition's correction history remains relevant. Three Google controls initially used numeric rather than documented decimal-string billing IDs. A byte-limit fixture was shortened by serialization, and an initial JavaScript runner changed large integers by parsing and reconstructing them. Corrected execution preserves raw strings. First-run artifacts remain available. These are harness corrections, not evidence that the product itself changed to satisfy those cases. Differential agreement is a consistency result because both interfaces share a core; it is not an independent semantic oracle.

5.6 Shuffled-block performance study

The added study uses the same 19 declared fixtures, ranging from one to 256 impressions, selected Google and Prebid cases, diagnostic-heavy input and a 64 KiB opaque extension. Each is measured in parse-only and selected-contract modes; four responses also receive paired validation. Each configuration is crossed with fixed input and 32 rotating auction-identity strings, producing 84 configurations. Rotation changes a small identity value and is deliberately a weak diversity check. It is not representative variation in message shape or a general cache-adversarial workload. Fixed and rotating strings can also differ slightly in byte length.

A seeded xorshift shuffle changes configuration order in each of 11 blocks. Each configuration receives 100 warmup calls, then at least 200 samples and 40 milliseconds of sampling, with a 100,000-sample cap. A per-call monotonic timer encloses parsing, validation, result allocation and destruction. It excludes process launch, I/O, network access and output serialization. Timer calibration is reported rather than subtracted. The original seven-trial measurements are retained as historical evidence; the revised headline uses the shuffled experiment.

Both environments use Rust 1.97.1 release builds of the frozen source. macOS arm64 runs on the Apple M4 workstation with 16 GiB RAM. Linux arm64 runs in its Colima Ubuntu virtual machine with

eight assigned CPUs and approximately 10 GiB RAM, using the pinned Rust container image. They share physical hardware and are not independent hardware replications. Runs are sequential; the pending study-owned Go build is paused during timing. The workstation remains shared, without isolated cores, controlled thermal state or a guarantee against unrelated activity.

We report the median of 11 block p95 estimates. Percentile bootstrap intervals resample whole block estimates 10,000 times with seed 20261110. Calls within a block are not resampled as independent observations. These intervals describe measured block variation under this schedule, not a confidence bound for all hardware or production p95 latency. Minimum and maximum block estimates remain visible to avoid concealing occasional scheduler effects.

6. Results

6.1 RQ1: failures, repair and the admission boundary

The frozen release agrees with 78 of 80 original binary expectations. The two disagreements are malformed embedded Native request strings. Both produce `openrtb.native.request.unparseable`, but at warning severity; therefore the public `valid` field remains true. This is a demonstrated distinction between detection and a blocking admission decision. The research wrapper's explicit strict-nested policy blocks those warnings without changing their original labels or misreporting the released core's behavior.

The seeded challenge produces a different, source-backed defect. The pinned IAB protobuf model declares `BidRequest.test` and `BidRequest.allimps` as booleans. The released divergence table omits them. In proto-json mode, the release accepts integer 0 and 1 and rejects boolean false and true at these two fields. Eight decisions therefore disagree with the declared representation contract. The remainder of the 792 reviewed cases agree with their expectations.

Adding the two fields to the sorted table changes all eight decisions as expected. The candidate agrees with all 792 reviewed challenge labels. Its existing 278 Rust tests pass after the public table-count assertion is changed from 28 to 30 and assertions for the two fields are added. Replay of all 3,761 existing cases preserves all 3,579 binary expectations and all 2,262 tested required or forbidden diagnostic assertions. The patch is a research candidate, not an unannounced new product release.

Table 2. Contract decisions and policy outcomes. Counts are test agreement, not estimated production accuracy.

Evidence	Frozen core	Candidate core	Candidate with strict policy
Original binary labels	78/80	78/80	80/80
Original scope controls	Boolean alone is insufficient	Boolean alone is insufficient	5/5 indeterminate
Reviewed seeded challenge	784/792	792/792	792/792
Existing binary regression labels	3,579/3,579	3,579/3,579	Not rescored as an independent oracle

The policy also rejects a response when its separately validated originating request is invalid. A response requiring paired admission becomes indeterminate if that request is absent. A supported violation can still justify rejection when some other declared scope is unavailable. These tests exercise policy precedence and evidence dependencies rather than merely duplicating a return-value assertion.

All 1,232 metamorphic boolean comparisons preserve the original decision. Each of five deliberately disabled paired-obligation families is distinguished by 128 challenge cases. This confirms that the challenge exercises those five families under its bounded construction. It does not establish exhaustive semantic coverage. The eight flag failures represent two missing field declarations, and the two malformed-Native rows repeat one severity mechanism. Reporting case counts must not turn those mechanisms into ten independent discoveries.

6.2 RQ2: the equal-context baseline changes the conclusion

Within the 77-case common scope, the independent local schema agrees with 44 labels. Canonical RTBLint agrees with 51, declared-dialect mode with 59, standalone selected-profile mode with 65, and full paired mode with 75. These ablations show what the supplied context and implemented obligations contribute. They do not separate context from mechanism.

The context-specialized reference agrees with all 77 labels. Candidate core alone still agrees with 75 because the Native-warning policy has not changed inside it. Candidate admission with strict-nested policy agrees with all 77. The correct inference is that the contextual contract and enforcement policy explain the observed distinction from the local schema. The data do not support a general claim that procedural rules detect obligations that an appropriately specialized schema-based system cannot detect.

Table 3. Common-scope comparison with equivalent input available to the contextual implementations.

Implementation or mode	Agrees	Eligible	Interpretation
Independent local schema	44	77	Local field description
Frozen canonical mode	51	77	Additional local semantics
Frozen declared-dialect mode	59	77	Selected representation
Frozen standalone profile mode	65	77	Selected integration rules
Frozen full mode	75	77	Originating request available
Context-specialized reference	77	77	Same context, separately implemented predicates
Candidate plus strict policy	77	77	Same context with explicit blocking policy

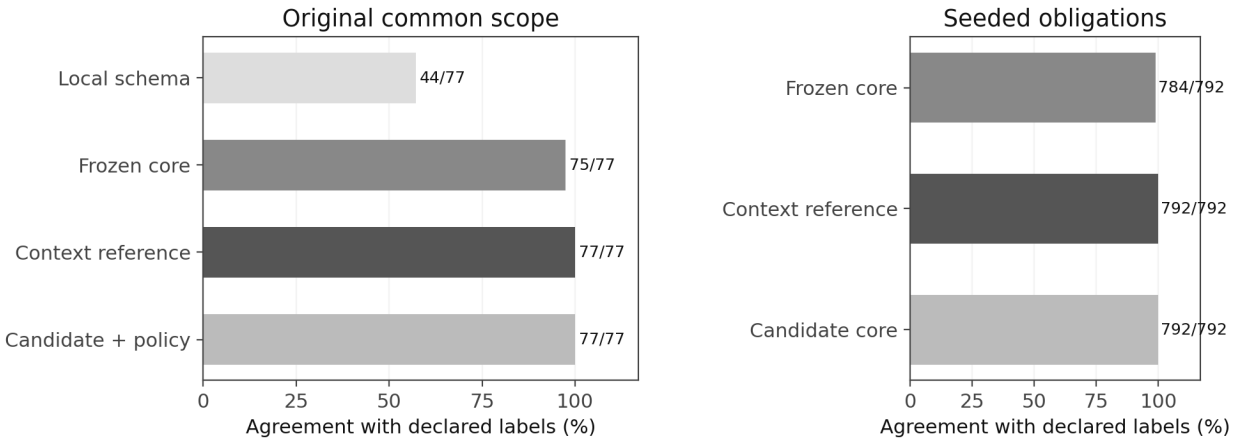


Figure 2. Matched-context and seeded comparisons. The original panel uses 77 common-scope labels; the challenge panel uses 792 author-derived labels. The reference receives equivalent context. Agreement on these directed cases is not a population accuracy estimate.

The separate reference also agrees with all 792 reviewed challenge expectations, while the frozen release misses the eight root-flag cases. Its value is therefore not only a benchmark score. It supplies an executable alternative interpretation that exposed a concrete omission and provided a repair target. Because the same author selected and interpreted the obligations, this remains weaker than blind external adjudication. The correction history is part of the result, not a reason to conceal the comparison.

6.3 RQ3: complete upstream receiver challenge

Across all 140 externally authored fixtures, frozen static validation agrees with 84 HTTP expectations. It accepts 27 of the 34 expected acceptances and rejects 57 of the 106 expected rejections. Seven expected acceptances are rejected, and 49 expected rejections receive no error-severity finding. The root-flag patch does not change this spec-json challenge. These counts refute treating a static pass as sufficient evidence of acceptance by the tested receiver.

Table 4. Static release decisions against upstream receiver expectations. Row labels are the maintainers' expected HTTP status, not universal protocol validity.

Upstream expectation	Static accept	Static reject	Total
HTTP 200	27	7	34
HTTP 400	49	57	106
Total	76	64	140

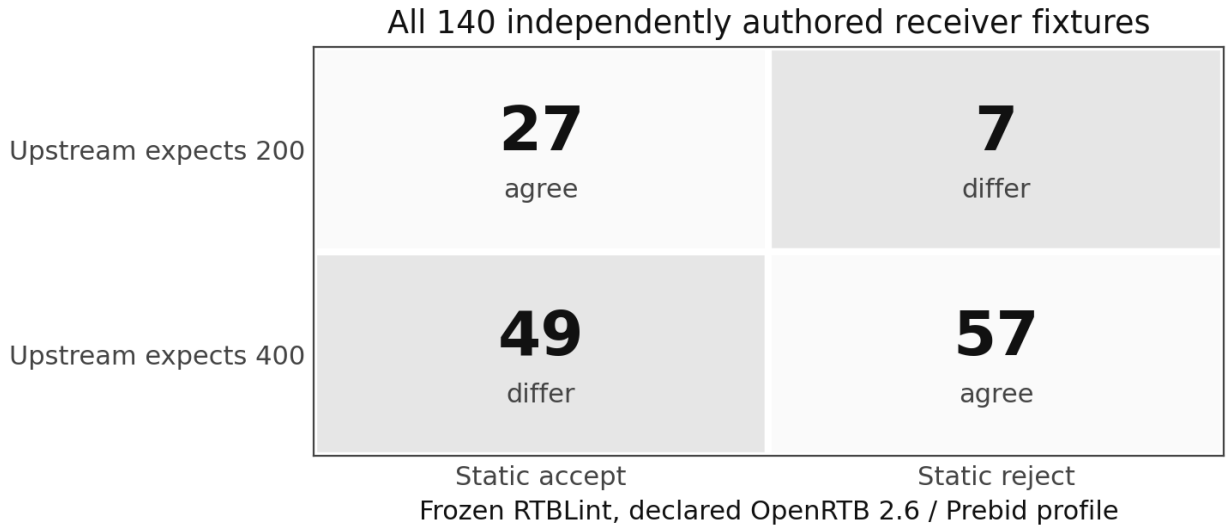


Figure 3. All 140 upstream fixtures, selected without using RTBLint outcomes. The 56 differing decisions remain in the artifact. Configuration, normalization and selected version are part of the receiver contract and cannot be inferred from static acceptance alone.

Six of the seven rejected positive fixtures produce relocated-field findings under the selected 2.6 contract. For example, `all-ext.json` carries `regs.ext.gdpr` and `user.ext.consent`; the checker points to `regs.gdpr` and `user.consent`. The set includes explicit 2.5/2.6 conversion fixtures, so imposing a normalized 2.6 contract at raw receiver ingress is an inappropriate claim of interchangeable boundaries. The remaining positive disagreement concerns an upstream SKAdNetwork extension fixture that omits fields required by the static extension rules. These facts do not justify simply weakening all of those checks. They identify which source contract, preprocessing step or fixture abstraction needs to be resolved.

The 49 accepted negative fixtures include categories the static profile does not comprehensively enforce: unknown bidder aliases and bidder parameters, receiver-specific size and positivity rules, duplicate impression identities, inventory-context requirements and detailed Native request constraints. Examples include `banner-w-negative.json`, `imp-id-duplicates.json` and `invalid-native/empty.json`. Some need only message-local logic; others depend on configured bidder names, stored requests or receiver policy. The complete fixture-level ledger includes upstream error text and RTBLint diagnostics so that subsequent work can adjudicate each obligation without treating all differences as equivalent.

These results are more useful as a boundary and coverage audit than as a product ranking. The profile derives from an earlier source boundary, fixtures exercise a specific receiver and configuration, and the challenge is not a random traffic sample. We do not call 84/140 an estimate of OpenRTB conformance accuracy, nor do we relabel the 56 disagreements to make it one.

6.4 Production-handler replay and normalized evidence

The production-handler experiment passes all 140 maintainer-authored status expectations: 34 responses are HTTP 200 and 106 are HTTP 400. Expected error substrings also pass the harness assertions. The executed test phase takes 13.746 seconds, excluding a lengthy cold build on the removable-drive source/cache volume. That duration is a test-run observation, not a server-throughput measurement. All 140 captured input digests verify. Replaying the actual captured strings preserves the static decision matrix in Table 4, including malformed outer request strings that cannot be represented as parsed request objects.

The experiment captures 34 normalized requests reaching the exchange, 34 successful response bodies and 43 outgoing generic mock-adapter requests. To examine normalization without changing the declared contract, we compare ingress and normalized requests under the same version, dialect and Prebid request profile. Passage rises from 27/34 at raw ingress to 32/34 after normalization. The canonical-stage control rises from 17/34 to 32/34. These are within-fixture observations of captured representations, not a population estimate or a claim that every preprocessing operation is correct.

Table 5. Request-stage comparison for the same 34 HTTP-200 fixtures. Both request stages use the same OpenRTB snapshot and dialect.

Capture stage	Canonical passage	Prebid-profile passage	Units
Original ingress string	17/34	27/34	Captured original request bytes
Normalized auction request	32/34	32/34	Captured request object, serialized for analysis

The trace for `all-ext.json` provides a concrete example. Its incoming `regs.ext.gdpr` becomes `regs.gdpr`; `user.ext.consent` becomes `user.consent`. The relocated-field errors disappear when the captured normalized representation is checked. A warning about the synthetic consent value can remain, illustrating that passage is still limited by the chosen severity policy. The normalized representation is an exchange-bound object serialized by the analysis, rather than a claim about bytes observed at a network boundary.

The two normalized rejections remain visible. `ortb-2.6-to-2.5-downconvert.json` retains `cattax: 20`, outside the static 2.6 taxonomy values; the fixture deliberately exercises conversion with synthetic values. `skadn.json` still lacks the version, source-app and network-list material required by the static extension rules. Both are accepted by the upstream handler in this controlled test. They warrant source and fixture-context adjudication, rather than being reported as newly proven server-conformance defects.

All 34 captured successful responses pass the core's paired-response check against the corresponding normalized request under the specification profile. That API result does not validate the originating request as an independent payload. Applying the admission wrapper with separate request validation yields 32 passages and two rejections, corresponding to the two rejected

originating requests. This is an empirical example of why a paired-response boolean alone is insufficient for a gate that promises validation of both messages.

The experiment establishes a practical boundary: contract selection and capture stage can alter a decision on the same receiver transaction. It also limits the result. Generic mock adapters and upstream test fetchers remain in use. The test does not establish transport conformance, real bidder interoperability, storage behavior, auction correctness or production acceptance rates. The complete traces permit further checks without concealing the receiver's observed acceptances or its synthetic fixture assumptions.

6.5 RQ4: interface consistency and regression scope

The release's native and distributed Node/WASM findings agree across all 3,846 original and committed cases when they receive identical raw strings and context. The comparison checks complete finding multisets, including identifiers, severities, paths, sections and messages. It is substantially stronger than comparing booleans alone. It still cannot detect a shared core defect, as the missing root flags illustrate. Candidate WASM equivalence is not claimed; the research repair and admission wrapper are evaluated natively.

Release replay matches every one of 3,579 validity expectations and all required/forbidden diagnostic assertions in the committed source-boundary manifests. The candidate preserves those results. This provides regression assurance for the measured change, while the external challenge demonstrates why regression success must not be presented as complete receiver coverage. Some expected labels came from the same implementation project being evaluated.

6.6 RQ4: shuffled-block processing cost

The additional timing study contains 9,061,291 calls on macOS and 9,221,270 on Linux, totaling 18,282,561. Each environment contributes 924 configuration-block observations: 84 configurations in 11 blocks. Timer calibration averages 15.64 ns on macOS and 15.71 ns on Linux. No calibration value is subtracted from the observations. The revised table uses medians of block percentile estimates rather than a pooled percentile over all calls.

Table 6. Selected fixed-input native costs in microseconds. Intervals are 95% percentile bootstrap intervals for the median block p95, resampling 11 blocks.

Workload	macOS p95	macOS interval	Linux p95	Linux interval
One request	2.13	1.75 to 2.17	1.92	1.92 to 1.92
One standalone response	2.13	2.04 to 2.38	2.04	1.92 to 2.13
One paired response	4.54	3.96 to 4.58	4.33	4.08 to 4.38
256-impression request	198.67	181.71 to 206.25	221.71	206.30 to 256.30
256-bid paired response	532.08	522.71 to 537.21	570.09	541.92 to 601.55
Prebid, 256 impressions	482.33	471.92 to 508.21	581.67	532.51 to 602.22
Opaque 64 KiB extension	10.00	8.29 to 10.38	11.04	10.63 to 11.21

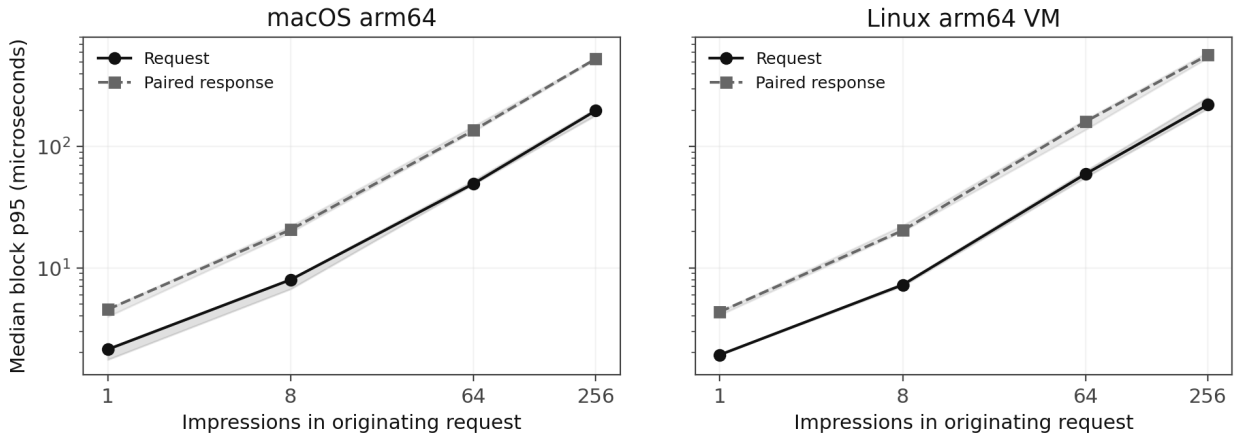


Figure 4. Median block p95 and block-bootstrap intervals for fixed request and paired-response workloads. Both environments run on the same physical M4. Logarithmic axes show structural scaling; shading describes observed block variation, not hardware-population uncertainty.

The small one-request interval on Linux partly reflects repeated values at the timer's effective resolution; it should not imply hundredth-microsecond hardware generality. Worst blocks are also materially larger than the medians for some large messages. The macOS 256-bid paired workload has a median block p95 of 532.08 microseconds but a largest block p95 of 1,127.21 microseconds. Its Linux counterpart has a largest block p95 of 704.51 microseconds. Google 256-impression validation reaches a largest macOS block p95 of 1,273.17 microseconds. The artifact preserves those observations rather than substituting the bootstrap interval for the observed range.

Across the 23 validation configurations, the rotating/fixed median-block-p95 ratio ranges from 0.932 to 1.099 on macOS and from 0.931 to 1.029 on Linux; the medians across configurations are 1.000 and 0.996 respectively. Changing auction identities did not reverse the measured scaling pattern. It

remains a narrow perturbation with largely repeated structure. It does not resolve the broader limitation of synthetic workloads or prove the absence of cache effects.

Structural work matters more than byte count alone. The opaque extension is much larger in bytes than the one-request fixture but cheap compared with traversing hundreds of impression or bid objects. Paired validation adds an originating-request parse and relationship checks, while profile-specific extension traversal can add work within each impression. These measurements support the practicality of the measured in-process API. They do not measure end-to-end server admission latency, network tail latency, sustained load, cold starts or the Python ledger wrapper, and they do not compare Rust execution speed fairly with the Python reference implementation.

7. Discussion

7.1 What the systems contribution establishes

The original local-schema contrast could encourage an overly strong conclusion: that an executable validator has an inherent detection advantage over schemas. The equal-context experiment does not support that conclusion. A separately specialized schema-based checker captures the tested request relationships and source-derived exceptions. RTBLint's engineering value is the packaged, versioned selection of those checks, its structured findings and its native execution cost. The comparison also shows how an alternative implementation can expose an omission in that package.

The root-flag repair is small, but its method matters. An independently maintained protobuf model defines a representation obligation; a bounded grid turns it into distinguishable cases; a separate checker disagrees with the release; the complete source table explains the failure; a minimal patch changes the predicted outcomes without disturbing the development replay. This is a reproducible defect-to-repair chain. It does not require claiming a general correctness guarantee from the patch's size or the challenge's perfect final score.

The Native warning example establishes a separate operational point. Evidence of malformed nested content can be present even when `valid` is true. A system integrator must choose whether that finding blocks admission. The research artifact makes that choice executable and visible in its policy record rather than reporting warning detection as if it already meant rejection.

7.2 A useful capture boundary is part of the contract

The Prebid challenge demonstrates why a company or project name alone cannot identify a sufficient validation contract. A raw publisher request, a normalized auction request and an outgoing bidder-specific request can have different applicable rules. A receiver can accept a legacy field at ingress, normalize it, and serialize another version for a bidder. Applying one normalized specification snapshot indiscriminately at all three points can produce misleading rejections.

An integration record should therefore identify the capture stage, actual version information when available, expected preprocessing and the direction of the selected profile. Original bytes remain available alongside any derived representation. The handler traces make these boundaries inspectable for one implementation. A validation system should not infer successful preprocessing solely because a receiver is known to support it in some configuration.

The negative upstream fixtures identify different work. Some accepted negatives need additional local constraints; duplicate impression IDs and invalid dimensions are examples. Others need the configured bidder inventory, alias map, account policy or stored-request data. A future capability declaration should distinguish these evidence requirements. Treating them all as ordinary schema omissions would hide which ones can be closed from the payload alone.

7.3 An actionable admission workflow

The artifact supports a concrete workflow. Preserve the incoming bytes, declare a contract and record the capture boundary. For a response, retain and independently validate its originating request if paired admission is required. Run the selected checker, retain every finding and apply a named policy. Emit an evidence ledger whose disposition is passage of selected checks, rejection or indeterminate. A decision can then be reproduced using the input digests, driver digest, policy source and archived context.

For example, a CI fixture containing a response with a syntactically valid but absent impression reference should fail the paired check. The same response without the required request should be indeterminate, because membership was not assessed. A malformed Native request is blocked under strict-nested policy even though the frozen core only warns. An unsupported profile direction is recorded as unsupported, not represented as a successful profile check. These outcomes correspond to different fixes: repair the message, supply evidence, or select and implement the appropriate contract.

The workflow is useful for fixture review, integration debugging and regression checks under a declared scope. The external results do not justify deploying the Prebid profile as an unconditional rejection filter at every ingress boundary. Before enforcing such a filter, users can inspect its decisions against captured receiver outcomes and resolve the identified normalization and coverage differences. The artifact provides the complete disagreement set needed for that assessment, rather than assuming that a static acceptance predicts receiver behavior.

The wrapper is a reference admission tool, not an optimized service. Its Python orchestration and process boundary are outside the Rust timing measurements. Integrators who need an in-process gate can implement the same policy over the core result. No measured latency is attributed to a wrapper that was not timed.

7.4 What a separate conformance study should add

A broader implementation study can reuse this artifact while asking a different question: which pinned servers honor which independently specified obligations at which boundary? It should freeze receiver versions and configuration, separate raw acceptance from normalized output and downstream behavior, and use runtime observations for timing, transport, auction and callback obligations. HTTP 200 alone does not establish that every required behavior occurred.

The present upstream challenge is a useful seed because its expectations were authored outside this study and the complete selected suites remain available. It covers one receiver with generic local mock adapters. Adding implementations, independent source adjudication and production-derived captures would strengthen external validity. Those extensions should be reported as new evidence rather than inferred from the current directed case counts.

8. Threats to validity and limits

8.1 Oracle and construct validity

The system maintainer authored the original contrast labels, generated challenge and context-specialization logic. Separate code and primary-source citations reduce shared implementation dependence but do not eliminate shared interpretation or confirmation bias. The external schema and Prebid expectations have third-party authorship; the added reference predicates do not acquire independent human adjudication merely by being written in Python. The paper discloses this distinction and preserves observed disagreements.

The final generated success is partly a repair evaluation. Feedback led to a corrected seat-list generator and then a source patch for the flagged dialect omission. It must not be read as untouched predictive validation. Five killed reference faults assess only five named relationships. Metamorphic invariance tests observable consistency, not semantic correctness. Native/WASM parity can preserve a shared defect. These evidence forms answer different questions and are not pooled into a single accuracy claim.

Receiver expectations are not a specification oracle. Configuration, normalization, deliberate receiver tolerance and synthetic fixture abstractions can explain differences. Conversely, an upstream rejection may identify a genuine missing static constraint. Complete rows and primary source anchors support later adjudication; the present experiment does not assert that every one of 56 disagreements has a unique, externally approved cause.

8.2 Internal validity and correction history

The study retains original and reviewed manifests, initial failure logs, exact source revisions and the candidate patch. Initial Google wire-shape errors, representation-changing harness behavior and the invalid seat-list generator show why this history is necessary. The final pipeline preserves input strings and hashes rather than relying on decoded-object identity. Hash equality proves byte consistency, not source interpretation correctness.

The Prebid harness reuses upstream construction helpers and fixture configuration, reducing the risk of accidentally comparing a different receiver contract. It still runs an in-process HTTP handler with test fetchers and generic adapters. It does not exercise every server startup option, commercial adapter or storage backend. The profile's earlier source boundary and the newer challenge commit are recorded rather than hidden by an unversioned project label.

8.3 Performance and external validity

Both runtime environments share the same physical M4. A VM boundary is an operating-environment check, not a second independent machine. Measurements use one thread on a shared workstation. Eleven shuffled blocks and block-level intervals reduce fixed-order and pseudoreplication problems but do not remove scheduler interference, temporal correlation, allocator effects or thermal drift. Some large configurations have only a few hundred samples per block, limiting precision of a tail estimate.

The rotating corpus changes only auction identities. It does not represent real-world diversity or adversarial structures. No network, end-to-end server throughput, cold start, concurrent-load behavior or Python admission latency is measured. The Rust and Python implementations are not used for a language-neutral speed ranking. Values should be treated as local cost observations under the archived workloads.

One server and selected profiles cannot establish ecosystem conformance. The challenge includes all fixtures in the declared directories, but the directories themselves are a purposive choice. Neither the receiver cases nor the directed protocol grid are probability samples of traffic. The data support reproducible counterexamples, targeted repair and engineering guidance within stated boundaries.

8.4 Artifact and source drift

The frozen release, candidate patch, external fixtures, source receipts, raw outcomes and report generators are versioned in the research package. Documentation URLs can change. Receipts identify retrieved bytes and retrieval scripts report mismatches. A source revision does not prove that every documented obligation was implemented. Publication should assign a persistent artifact identifier and permit independent replay; no such deposit or peer review is claimed here.

9. Conclusion

RTBLint makes specification revision, representation, integration direction and paired evidence explicit in an executable validation system. The revised evaluation shows why those dimensions matter and also limits the claims that can be made from them. An equal-context schema-based reference matches the tested obligations and exposes a missing root-level dialect mapping. A small candidate repair removes the demonstrated failures, and an explicit admission policy turns a nested-content warning into a reviewable blocking decision without changing the released core's reported behavior.

The independent receiver challenge is the stronger practical caution: static decisions differ from 56 of 140 upstream HTTP expectations under the selected capture contract. Complete handler traces and disagreement records allow those boundaries to be investigated instead of treating a static pass as receiver approval. Byte-identical interface checks, existing regression replay and shuffled-block native timings provide complementary implementation evidence. Together, the results support versioned, evidence-preserving admission as a useful systems primitive whose scope remains visible to its users.

Data and artifact availability

The self-contained HTML and PDF are generated from this revision's `manuscript.md`. The research package retains first-edition evidence and places the added protocol, comparison code, case manifests, source patch, traces, admission wrapper and raw block observations under `v2/`. The artifact README gives exact reproduction commands. Input and source digests identify the recorded versions. A local frozen protocol is provided; it was not externally preregistered.

The shareable package includes the Apache-licensed system archive, IAB protobuf source and applicable license notices, the MIT-licensed external schema, and the pinned Prebid fixture archive with its license. Other official documentation snapshots remain available through retrieval receipts. This is an author-review preprint without a public archival DOI, submission or external human adjudication.

Competing interests and use of generative AI

The author maintains RTBLint and related validation tools and authored earlier OpenRTB preprints. This interest can affect rule selection and interpretation. OpenAI Codex assisted with experiment implementation, manuscript development and artifact preparation. Execution records support the reported numerical outcomes. Neither automated assistance nor a separate checker is represented as independent human review. Scientific interpretations, source obligations and deployment decisions remain matters for author and reviewer assessment.

References

[1] IAB Tech Lab. *OpenRTB 2.6 Specification and Implementation Notes*, release 2.6-202606. June 2026. https://github.com/InteractiveAdvertisingBureau/openrtb2.x/tree/2.6-202606

[2] IAB Tech Lab. *OpenRTB Protocol Buffers Schema, v2*. Source snapshot and SHA-256 receipt archived with this paper. https://github.com/IABTechLab/openrtb-proto-v2

[3] Protocol Buffers. *ProtoJSON Format*. Documentation retrieved November 10, 2026. https://protobuf.dev/programming-guides/json/

[4] Prebid.org. *Prebid Server: /openrtb2/auction*. Documentation retrieved November 10, 2026. https://docs.prebid.org/prebid-server/endpoints/openrtb2/pbs-endpoint-auction.html

[5] Sekowski, A. *How Machine-Checkable Is OpenRTB? Classifying the Normative Content of the Protocol That Clears Real-Time Advertising*. Preprint, July 2026. https://doi.org/10.13140/RG.2.2.27937.57448

[6] Sekowski, A. *Measuring OpenRTB Dialects in Client-Side Header Bidding*. Preprint, revised August 2026. https://doi.org/10.13140/RG.2.2.26572.78720

[7] Andrews, H., Wright, A., Hutton, B., and Dennis, G. *JSON Schema: A Media Type for Describing JSON Documents*. Draft 2020-12. https://json-schema.org/draft/2020-12/json-schema-core

[8] Wright, A., Andrews, H., Hutton, B., and Dennis, G. *JSON Schema Validation: A Vocabulary for Structural Validation of JSON*. Draft 2020-12. https://json-schema.org/draft/2020-12/json-schema-validation

[9] Attouche, L., Baazizi, M.-A., Colazzo, D., Ghelli, G., Sartiani, C., and Scherzinger, S. *Validation of Modern JSON Schema: Formalization and Complexity*. arXiv:2307.10034v2, 2024. https://arxiv.org/abs/2307.10034

[10] Viotti, J. C., and Mior, M. J. *Blaze: Compiling JSON Schema for 10x Faster Validation*. arXiv:2503.02770v2, 2025. https://arxiv.org/abs/2503.02770

[11] OpenRTB community. *openrtb/openrtb2x: reference implementation and openrtb-validator*. Repository documentation retrieved November 10, 2026. https://github.com/openrtb/openrtb2x

[12] Silveira, K. *OpenRTB 2.6 JSON Schema*. Commit b81b8acf71349ed60333b370517bf417fdace12b, 2025. https://github.com/klaussilveira/openrtb-json-schema/tree/b81b8acf71349ed60333b370517bf417fdace12b

[13] Google. *OpenRTB Integration*. Authorized Buyers documentation retrieved November 10, 2026. https://developers.google.com/authorized-buyers/rtb/openrtb-guide

[14] Chaudhary, P. *Do Open-Source Video Ad Clients Follow VAST 4.3? An Empirical Study of Player Conformance*. arXiv:2610.39093, 2026. https://arxiv.org/abs/2610.39093

[15] Kallus, B., Anantharaman, P., Locasto, M., and Smith, S. W. *The HTTP Garden: Discovering Parsing Vulnerabilities in HTTP/1.1 Implementations by Differential Fuzzing of Request Streams*. arXiv:2405.17737, 2024. https://arxiv.org/abs/2405.17737

[16] Thomson, M., and Schinazi, D. *Maintaining Robust Protocols*. RFC 9413, Informational, June 2023. https://www.rfc-editor.org/rfc/rfc9413.html

[17] IAB Tech Lab. *OpenRTB Dynamic Native Ads API Specification, Version 1.2*. 2017. https://iabtechlab.com/wp-content/uploads/2016/07/OpenRTB-Native-Ads-Specification-Final-1.2.pdf

[18] Sekowski, A. *RTBLint*. Version 0.14.7, commit 4defcb383729e20147a82ca5737974be03426192, 2026. https://github.com/aleksUIX/rtblint/tree/4defcb383729e20147a82ca5737974be03426192

[19] Sekowski, A. *VAST XML Validation at Bid-Time Scale: Latency Analysis and Integration Patterns for Programmatic Video Pipelines*. Preprint, April 2026. https://doi.org/10.13140/RG.2.2.11404.27520

[20] Prebid.org contributors. *Prebid Server*, commit 46670d63b5b749cb4d52eb0d2b0a801ee28c7721. Upstream endpoint fixtures and test infrastructure. https://github.com/prebid/prebid-server/tree/46670d63b5b749cb4d52eb0d2b0a801ee28c7721

Appendix A. Source and implementation map

The following paths refer to the frozen source archive. They locate implementation evidence, not independent proof that every source interpretation is correct.

Component	Source path	Observable responsibility
Public results and APIs	`crates/rtblint-core/src/lib.rs`	Context entry points, issue fields, boolean result
Compiled catalog generation	`crates/rtblint-core/build.rs`	Static shapes, fields and citation metadata
Catalog access	`crates/rtblint-core/src/canonical_catalog.rs`	Versioned object and field lookup
Local traversal	`crates/rtblint-core/src/validator.rs`	Parse, root, shape, local and dispatch behavior
Encoding	`crates/rtblint-core/src/dialect.rs`	Declared dialect and 28-field divergence table
Profiles	`crates/rtblint-core/src/profile.rs` and `src/profile/`	Direction support, exceptions and documented conditions
Pairing	`crates/rtblint-core/src/pair.rs`	Originating-request context and response relationships
Native content	`crates/rtblint-core/src/native.rs`	Embedded JSON and selected asset constraints
Node interface	`npm/index.cjs` and `npm/wasm/`	Distributed WASM calls used by the parity runner

Appendix B. Reproduction sequence

Use a fresh extracted package. Install the pinned Python requirements from `evaluation/requirements.txt`, Rust and Node for the first-edition replay, and Go 1.25 or newer for the receiver experiment. The performed host receiver run uses Go 1.26.5. A compatible `patch` executable is required for the candidate source. The added replay command is:

```
python v2/evaluation/reproduce_v2.py \
--workdir /tmp/rtblint-reproduction
```

The runner verifies the frozen archive, creates separate frozen and candidate source trees, applies the published patch and compiles both with committed driver lockfiles. It reruns the original and generated decisions, contextual reference comparison, metamorphic checks, reference-fault sensitivity and admission ledgers. An optional `--pbs-checkout` argument uses a fresh checkout at the

exact pinned Prebid commit, installs only the research `_test.go` recorder and executes the upstream production handler with local adapters. Normal initial dependency retrieval may require network access. Experiment execution needs no production account, remote bidder or live advertising traffic.

`check_candidate_regression.py` reruns the complete committed corpus against the repaired candidate. The first-edition runner and native/WASM records remain under `evaluation/` and `results/`. The candidate changes only native validation in this study; do not interpret the released WASM record as a test of a rebuilt candidate WASM module.

The shuffled benchmark driver source and its lockfile are under `v2/evaluation/benchmark-driver/`. Compile it against the extracted frozen source and invoke `bench-blocks` with `evaluation/benchmark-cases.json`. Raw observations remain separate for macOS and Linux. `summarize_blocks.py` and `make_figures_v2.py` reproduce block summaries and figures from the archived records. Recollecting timings can change the estimates; it should create new observations rather than silently replace the publication record.

For a response gate, the executable interface is:

```
python v2/evaluation/admit.py response.json \
--role response --request request.json \
--require-pair --profile spec \
--driver /tmp/rtblint-reproduction/candidate-target/release/rtblint-paper-driver \
--output admission.json
```

The default policy is strict-nested. Exit 0 means passage of the selected checks, 2 rejection, 3 indeterminate and 4 execution failure. The JSON ledger records original UTF-8 hashes, contract context, driver and policy-source hashes, findings and unobserved domains. A different capture boundary or policy must be recorded rather than inferred from the same filename.

Appendix C. Corrections and complete disagreements

The artifact retains the initial and reviewed original manifests, the first representation-changing executions, the initial generated seat-list context, and initial reference-checker outcomes. Corrections have explicit source rationales and preserve labels. `root-proto-flags.patch` includes both the source change and the inventory assertions; `candidate-provenance.json` identifies its base and digest.

The external fixture archive includes the pinned upstream license. `upstream-cases.jsonl` preserves every selected expectation. The handler trace file records every actual status, body, normalized request and available mock-adapter request. `upstream-comparison.jsonl` retains every static result, including the 56 differences. A separate integration summary checks actual captured bytes and outcomes. A 101-case blinded-input packet is prepared for external source adjudication; no human review has been performed. There is no outcome-based deletion or relabeling of these external cases.

The case ledgers are intended to support future independent review. A reviewer can start with a single reported mismatch, recover the exact source fixture and selected contract, execute it in both implementations, inspect its handler trace and then decide whether the appropriate remedy is a validator rule, a new profile boundary, additional evidence or a documented receiver tolerance.