Convert Once, Consume Many: SDF for Cacheable, Typed Semantic Extraction from Web Pages

Version 0.2.0 — February 2026

Pranab Sarkar

Independent Researcher

developer@pranab.co.in

https://github.com/spranab

Project: https://sdfprotocol.org | https://github.com/sdfprotocol

DOI: https://doi.org/10.5281/zenodo.18559223

Abstract

AI agents and large language models (LLMs) typically consume web pages by fetching HTML and repeatedly performing boilerplate removal, chunking, and semantic extraction (entities, claims, and task-specific structure) at query time. This "re-extract per consumer" pattern increases latency and inference cost, and yields inconsistent outputs across models and prompts. We present **Structured Data Format (SDF)**, an open, schema-validated JSON protocol for publishing or serving **pre-extracted, agent-oriented semantic representations** of web content. SDF specifies (i) a hierarchical content type system with type-specific `type_data` and optional multi-type overlays ("aspects"), (ii) a conversion pipeline that separates type classification from type-conditioned extraction, (iii) explicit provenance (converter/model identifiers, processing chain, and source-content hashing) and optional temporal freshness signals, and (iv) content negotiation for multiple resolution levels.

We report the first production-scale deployment of SDF: a crawler processed **6,206 URLs** and generated **2,335+ schema-valid SDF documents** across **10 parent types** and **74 observed type combinations**, with a **~96% pipeline success rate** on successfully fetched content (most failures were fetch-level: connection errors, rate limiting, dead links). A two-pass, locally hosted pipeline — consisting of a QLoRA-fine-tuned **1.5B classifier** (Qwen2.5-1.5B-Instruct) and a fine-tuned **3B extractor** (SmolLM3-3B) — achieves a **4.1x latency reduction** relative to a 14B baseline while retaining **90% exact extraction accuracy** on a held-out benchmark and **100% JSON validity** in production. Telemetry across the production corpus revealed substantial taxonomy drift, with the classifier inventing 63 type combinations not present in the protocol taxonomy; a deterministic five-stage **type normalization cascade** corrects these violations and guarantees taxonomy compliance before extraction. End-to-end conversion on consumer hardware (dual RTX 3090 Ti) averaged **~18s/document** with the fine-tuned pipeline, after which cached SDF responses can be served in **<10ms**. SDF payloads averaged **1.8KB (compact)** to **4.2KB (full)** versus **~89KB** raw HTML (uncompressed).

A downstream consumption experiment compared general-purpose **7B and 3B models** answering questions from raw markdown versus pre-extracted SDF across 30 documents and 5 question types. The SDF path achieved **0.739 mean accuracy** versus **0.352 raw** at 7B ($p < 0.05$, $t(29) = 11.890$) and **0.606 versus 0.333** at 3B ($p < 0.05$, $t(29) = 9.665$), with a **58.5% latency reduction** and **99.2% token reduction** relative to raw HTML (103,013 → 834 estimated tokens) — providing the first empirical evidence that SDF's "convert once, consume many" value proposition generalizes across model scales.

SDF does not propose new extraction models; its contribution is a **standardized, verifiable-by-validation interchange layer** that amortizes extraction and makes downstream agent consumption cheaper, more uniform, and auditable.

Keywords: structured data extraction, web content representation, AI agents, semantic web, JSON schema, content type classification, information extraction, large language models

1. Introduction

AI agents are turning the web into an on-demand knowledge substrate, but today's agent pipelines repeatedly pay the same "HTML tax": every consumer refetches pages, strips boilerplate, chunks text, and re-extracts entities and claims — often with inconsistent results and no auditable provenance. This paper argues that the missing layer is not another extractor but a **standardized semantic intermediate representation**: a cacheable, schema-validated, typed, and provenance-linked snapshot of a page's agent-relevant semantics.

1.1 The Problem: Redundant Semantic Inference

Modern AI agents interact with web content through a pipeline that has remained largely unchanged since the early days of web scraping:

1. **Fetch** raw HTML from a URL
2. **Strip** tags and boilerplate to extract readable text
3. **Tokenize** the full text into the agent's context window
4. **Infer** entities, topics, claims, and relationships
5. **Rebuild** context and structured understanding from scratch

When thousands of agents process the same URL, each independently repeats this entire pipeline. A popular news article might be fetched, parsed, and semantically analyzed millions of times by different agents, each producing slightly different extractions depending on their model, prompt, and context window.

This pattern mirrors long-studied web extraction pipelines in information retrieval and wrapper induction [13, 14], where repeated boilerplate removal and content parsing is a dominant cost. Recent agentic and RAG-based systems [9, 17] further amplify this cost by repeatedly chunking, embedding, and re-extracting structure per consumer rather than amortizing it across consumers.

This redundancy has concrete costs:

- **Compute**: Each extraction consumes LLM inference tokens proportional to the full document length
- **Latency**: Real-time agent workflows are bottlenecked by sequential fetch-parse-infer pipelines

- **Inconsistency**: Different agents extract different entities, relationships, and claims from the same content
- **Bandwidth**: Raw HTML is 3–10x larger than the agent-relevant semantic payload
- **Freshness blindness**: Agents have no signal about whether extracted facts are current, stale, or time-sensitive

1.2 The Solution: Pre-Compiled Semantic Content

SDF addresses this by defining a **canonical format for pre-compiled, semantic web content** that agents can consume directly. It is the missing layer between raw web content and agent reasoning — analogous to an intermediate representation (IR) that makes downstream consumption cheaper and more uniform: the "compilation" step is semantic extraction into a schema with explicit provenance, rather than executable transformation.

1.3 Design Principles

Principle	Description
Schema-deterministic	For a fixed converter version, prompt/template version, and model configuration, the converter produces schema-valid outputs with stable field structure. Provenance records the configuration needed to reproduce results.
Schema-validated	Every SDF document conforms to a modular JSON Schema (draft 2020-12)
LLM-agnostic	Not tied to any specific model, provider, or inference runtime
Cacheable	SHA-256 content hashes enable deduplication and efficient caching
Human-inspectable	Plain JSON format readable by both humans and machines
Type-aware	Hierarchical type system with type-specific structured extraction
Auditable	Provenance chains record converter, model, and processing steps for reproducibility and trust policies
Extensible	Vendor-namespaced extensions without polluting the core schema
Versioned	Protocol header with feature flags, deprecation tracking, and dialect negotiation
Gracefully degradable	Partial extraction from smaller models still produces valid SDF documents

Determinism note. Because SDF conversion may use probabilistic models, SDF targets determinism at the **interface level** (field presence, types, constraints, and validation) and reproducibility via explicit provenance (converter/prompt/model identifiers). Consumers should treat content hashes as hashes of the **source content**, not guarantees of identical extracted fields across converters.

1.4 Contributions

This paper contributes:

1. **A protocol and schema suite** for representing web documents as agent-oriented semantic JSON with hierarchical typing, type-specific `type_data`, and explicit provenance metadata.
2. **A conversion architecture** that separates type classification from type-conditioned extraction, with a deterministic type normalization cascade that corrects taxonomy drift from probabilistic classifiers.
3. **A fine-tuned dual-model pipeline** demonstrating that QLoRA-fine-tuned 1.5B and 3B models can achieve 90% exact match accuracy at 4.1x lower latency than a 14B baseline, with a combined model footprint under 3 GB.
4. **A production-scale empirical study** over 2,335 documents across 10 parent types, characterizing type distribution, vocabulary drift, extraction quality, and operational performance alongside a controlled benchmark comparing four model configurations.
5. **A downstream consumption experiment** demonstrating that general-purpose 7B and 3B models both answer questions significantly more accurately (0.739 vs 0.352 at 7B; 0.606 vs 0.333 at 3B) and 58.5% faster when consuming pre-extracted SDF versus raw markdown, with 99.2% token reduction relative to raw HTML — providing the first empirical support that the "convert once, consume many" value proposition generalizes across model scales.

We explicitly do **not** claim new information extraction algorithms; SDF is a systems protocol intended to standardize and amortize extraction.

Paper roadmap. Section 2 situates SDF relative to web structured data, extraction, and provenance standards. Section 3 formalizes the content structuring problem. Section 4 defines the SDF document model and protocol semantics. Section 5 describes the two-pass conversion pipeline. Section 6 details the modular schema system. Section 7 evaluates classification, extraction coverage, runtime cost, and ablations. Section 8 covers batch processing. Section 9 discusses deployment trade-offs and security considerations. Section 10 outlines future work.

---

2. Related Work

SDF sits at the intersection of (i) web structured data standards, (ii) web content extraction and archiving, (iii) information extraction and knowledge representation, and (iv) modern agent/RAG tooling that ingests web content. This section positions SDF relative to each and clarifies what SDF standardizes (an interchange format + protocol semantics), versus what it intentionally does not (a novel extraction model).

2.1 Structured Data Standards on the Web

Schema.org [1] defines a publisher-authored vocabulary commonly embedded via JSON-LD, Microdata, or RDFa. In practice, Schema.org coverage is inconsistent across sites and often incomplete for agent needs (e.g., claims, section structure, or type-specific fields like API endpoints). SDF is complementary: it can incorporate Schema.org fields when available, but it targets a **converter-generated**, schema-validated representation that is consistent even when publisher markup is absent or low quality.

Open Graph Protocol [2] and **Twitter Cards** provide lightweight metadata (title, description, image) for social sharing. They are useful as inputs to SDF's `source` metadata but do not attempt semantic extraction.

JSON-LD [3] is a linked-data serialization with RDF semantics. SDF prioritizes **agent consumption efficiency and validation** (fixed JSON Schema constraints, resolution levels, and provenance fields) rather than RDF graph interoperability. A future profile could define a normative JSON-LD context for SDF.

2.2 Web Content Extraction and Boilerplate Removal

Boilerplate removal and main-content extraction are well-studied in web IR. Tools such as **Mozilla Readability** [4], Boilerpipe [13], jusText, and trafilatura remove navigation, ads, and template noise to recover article-like content. SDF uses such extractors as a preprocessing step but differs in scope: it standardizes a **post-extraction semantic packaging** (entities, claims, relationships, type-specific fields) plus provenance and negotiation semantics.

2.3 Web Archiving, WARC, and Large-Scale Web Corpora

WARC (Web ARChive format) [18] is the de facto standard for storing crawled web resources (HTTP request/response payloads and metadata). WARC excels at preserving raw fetch artifacts for replay and reproducibility, but it does not standardize semantic fields useful for agents (claims, entities, type-specific structure). SDF is complementary: a natural deployment is to store WARC for raw preservation and SDF as a derived, validated semantic layer with explicit provenance linking back to the archived payload hash.

Common Crawl [19] provides web-scale snapshots for research and retrieval. Common Crawl-style pipelines typically require repeated downstream parsing, boilerplate removal, and extraction to build usable datasets. SDF can be viewed as an intermediate representation produced alongside crawls, enabling more consistent downstream processing.

2.4 Commercial and Hosted Web-to-Structured APIs

Several commercial systems provide structured extraction from arbitrary web pages, most prominently **Diffbot**, which converts pages into structured entities and knowledge graph entries. The key difference is **standardization and interoperability**: SDF specifies an open schema and protocol semantics (typing, provenance, negotiation), and can serve as a target interchange format for multiple extractors — commercial or open-source — rather than binding consumers to a single vendor's schema.

2.5 Reader Services and Agent-Focused Web Fetching

Recent agent ecosystems commonly rely on "reader" endpoints that transform HTML into cleaner text/markdown for LLM ingestion, e.g., **Jina Reader**, **FireCrawl**, and similar services. These systems primarily standardize content retrieval and cleaning (HTML to text/markdown, optional metadata). SDF differs by standardizing a **semantic output contract**: typed `type_data`, extracted `entities`, `claims`, `relationships`, temporal freshness signals, and provenance fields validated by JSON Schema. Reader services reduce input noise; SDF reduces repeated semantic inference by publishing a structured representation.

2.6 Agent/RAG Framework Ingestion Pipelines

Frameworks such as **LangChain** (document loaders) and **LlamaIndex** provide connectors that fetch web pages, convert them to text, split into chunks, and build vector indexes for retrieval-augmented generation [9]. These pipelines operationalize ingestion, but intermediate representations are typically framework-internal and do not standardize cross-system interchange of extracted semantic fields such as claims with temporal sensitivity. SDF can serve as a stable, cacheable document representation that loaders ingest directly.

2.7 Information Extraction, OpenIE/KBP, and Claim Verification

SDF's `entities`, `relationships`, and `claims` fields overlap with classical information extraction (IE), open information extraction (OpenIE) [14], and knowledge base population (KBP). Prior work focuses on extraction models and evaluation protocols; SDF instead specifies **how extraction artifacts are packaged, typed, validated, and attributed** for downstream agent consumption. Similarly, claim verification research (e.g., FEVER [15]) focuses on assessing truthfulness against evidence; SDF's `verification` metadata is an interface hook for downstream verification systems rather than a claim that SDF itself guarantees truth.

2.8 Provenance and Trust

Provenance standards such as **W3C PROV** [16] and related work on signed assertions (e.g., nanopublications [20], verifiable credentials) formalize derivation and trust chains. SDF borrows these motivations but adopts a pragmatic JSON envelope designed for low-friction deployment. Future versions could define normative mappings from `provenance.chain` to PROV-O and specify canonicalization and signature profiles for attestations.

We emphasize peer-reviewed sources where available; vendor documentation is cited only to describe specific API mechanisms rather than as scientific evidence.

---

3. Formal Problem Definition

This section formalizes the content structuring problem that SDF targets. The goal is not to define a single "correct" extraction, but to define a **standardized, validated representation** that is (i) useful for agents, (ii) auditable, and (iii) cacheable and interoperable across producers/consumers.

3.1 Inputs and Outputs

Let a web resource be identified by a URL u. Fetching u at time t yields an HTTP response containing a byte string $h(u,t)$ (typically HTML) and headers $H(u,t)$.

A preprocessing function pi maps the raw response to a normalized textual representation $x(u,t)$ (e.g., cleaned Markdown) plus extracted basic metadata (title, author, timestamps). In practice, pi implements boilerplate removal and HTML-to-text normalization.

We define a **content hash** function $hash(x(u,t))$ used for caching and deduplication. This hash is computed over the normalized content, not over the extracted semantic fields.

The converter produces an SDF document *d(u,t)* as: *f_theta(x(u,t)) = d(u,t)*, where *f_theta* is a parameterized conversion function (e.g., an LLM-based pipeline) with configuration *theta* that includes converter version, prompt templates, model identifier, and decoding settings.

3.2 SDF as a Typed, Validated Semantic Envelope

An SDF document *d* is a JSON object constrained by a JSON Schema *S_v* for protocol version *v*. SDF defines:

- A **type assignment** *tau(d) = (parent_type, type)* where `parent_type` is drawn from a closed set and `type` is validated conditionally on `parent_type`.
- A **type-specific data object** `type_data(d)` whose schema depends on *tau(d)*.
- A **semantic core** consisting of summaries, entities, claims, relationships, topics, and sections.
- A **provenance record** *prov(d)* that includes at minimum the converter identifier, model identifier, and a content hash linking *d* to *x(u,t)*.

SDF targets the following interface-level invariants:

1. **Schema validity:** *Validate(d, S_v) = true*
2. **Type-aware constraints:** *Validate(type_data(d), S_tau(d)) = true*, where *S_tau(d)* is the leaf schema for the assigned type
3. **Auditability:** *d* contains sufficient provenance to identify *theta* and to link back to the source content hash

3.3 Optimization Objective (Convert Once, Consume Many)

Let *A* be a set of agents that need to answer tasks *q* using web content. In a conventional pipeline, each agent incurs a per-consumption cost:

C_raw(u,q) = C_fetch(u) + C_parse(u) + C_infer(u,q)

where *C_infer* includes LLM token costs for extraction/reasoning over raw text.

Under SDF, a producer pays a one-time (per content hash) conversion cost:

C_conv(u,t) = C_fetch(u) + C_parse(u) + C_extract(x(u,t))

and each consumer pays a reduced per-consumption cost:

C_sdf(u,q) = C_fetch_sdf(u) + C_consume(d(u,t), q)

where *C_consume* is typically smaller because the agent can route on types, read summaries/claims directly, and avoid re-running extraction.

If a content hash is consumed *k* times, the amortized cost under SDF is *C_conv + k* C_sdf*, compared to* k * C_raw. *The protocol's purpose is to make* C_sdf* small and predictable by standardizing the representation and validation constraints.

3.4 Scope and Non-Goals

SDF does not define a unique "correct" semantic parse of a document. Instead, it defines: - A **stable contract** for what fields may exist and how they are typed - A **validation mechanism** for

structural correctness - A **provenance interface** so consumers can apply trust and reproducibility policies

Truthfulness of claims with respect to the world is out of scope; SDF primarily targets faithfulness to the source document and auditable conversion.

---

4. Protocol Architecture

4.1 Document Structure

An SDF document is a self-contained JSON object with the following top-level structure:

```
┌─────────────────────────────────────────┐
│ SDF Document v0.2.0                     │
├─────────────────────────────────────────┤
│ Protocol Header (sdf)                   │
│   version, schema_uri, dialect, features│
├─────────────────────────────────────────┤
│ Identity                                │
│   id (sdf:<sha256>), canonical_url      │
├─────────────────────────────────────────┤
│ Type Classification                     │
│   parent_type, type, type_data, aspects[]│
├─────────────────────────────────────────┤
│ Source Metadata                         │
│   url, title, author, site_name,        │
│   published_at, fetched_at              │
├─────────────────────────────────────────┤
│ Semantic Core                           │
│   summary, entities[], claims[],        │
│   relationships[], topics[], sections[] │
├─────────────────────────────────────────┤
│ Temporal Signals                        │
│   published_at, retrieved_at, freshness │
├─────────────────────────────────────────┤
│ Provenance                              │
│   converter, model_used, content_hash,  │
│   chain[], attestations[]               │
├─────────────────────────────────────────┤
│ Extensions                              │
│   links[], embeddings[], profiles,      │
│   extensions (vendor-namespaced)        │
└─────────────────────────────────────────┘
```

Conformance note. A producer is conformant if it emits JSON that validates against the SDF root schema for the declared `sdf_version`. Optional capabilities (e.g., `temporal`, `links`, `embeddings`, `profiles`) are advertised via the `sdf.features` list; consumers MUST ignore unknown fields and features. Version negotiation is handled at the HTTP layer via content negotiation and by schema

URIs embedded in the `sdf` header.

Normative vs. optional features. In v0.2.0, the normative core comprises the document envelope, hierarchical typing, `summary`, `source`, and `provenance` fields, plus schema validation rules. Features such as `embeddings`, `profiles`, and cryptographic `attestations` are optional extensions; this paper describes their intended use but does not require implementations to support them for conformance.

4.2 Hierarchical Type System

SDF v0.2 introduces a two-level hierarchical type system inspired by MIME types. Every document has two type fields:

- **`parent_type`**: One of 10 broad content categories (enumerated, schema-enforced)
- **`type`**: A specific content subtype (string, validated per parent)

Together they form a type path like `article.news` or `documentation.api_docs`.

Taxonomy stability. The type taxonomy is versioned; this paper reports the v0.2.0 taxonomy used by `sdf-engine/0.2.0`. All evaluation labels in Section 7 correspond to leaf types defined in this taxonomy.

Type Taxonomy (v0.2.0)

```
article                  documentation            commerce
├── news           *     ├── api_docs      *      ├── product       *
├── blog                 ├── support       *      ├── job_posting
├── opinion              ├── tutorial             ├── real_estate
├── review         *     ├── guide                ├── service
├── analysis             ├── changelog            └── auction
└── press_release        └── faq

data                     reference                discussion
├── finance        *     ├── encyclopedia         ├── forum_thread
├── sports               ├── academic_paper       ├── q_and_a       *
├── weather              ├── legal_document       ├── social_post
└── scientific_dataset   ├── recipe               └── comment_thread
                         ├── medical_info
                         └── patent

code                     media                    profile
├── repository     *     ├── video                ├── person
├── package              ├── podcast              ├── organization  *
├── code_snippet         ├── image_gallery        └── place
└── pull_request         └── music_album

event
├── conference     *
├── meetup
├── webinar
```

```
├── live_event
└── concert
```

Types marked with * are the 10 MVP types with full type-specific extraction prompts and JSON Schema validation.

Design Rationale The two-field approach was chosen for several reasons:

1. **No string parsing required**: Consumers can route on `parent_type` without splitting
2. **Graceful fallback**: Unknown leaf types can still be processed using known parent type semantics
3. **Schema enforcement**: `parent_type` is a closed enum; `type` is validated per-parent
4. **MIME precedent**: Mirrors the `type/subtype` pattern familiar to web developers
5. **Indexing**: Both fields can be independently indexed for efficient filtering

Type-Specific Data (`type_data`) Each content type defines a `type_data` object with fields specific to that content category. The schema uses **flat merged inheritance** — parent base fields and leaf-specific fields coexist in one flat object.

For example, `documentation.api_docs` extracts: - **Parent fields** (documentation): `product_or_system`, `audience`, `prerequisites`, `difficulty`, `code_examples` - **Leaf fields** (api_docs): `api_name`, `endpoints[]`, `base_urls`, `auth_methods`, `rate_limits`, `pagination`, `error_model`

Multi-Type Content: Aspects Real-world content often spans multiple types. A Bloomberg earnings article is primarily `article.news` but also contains financial data. SDF handles this through **aspects** — typed overlays that attach secondary type classifications with their own `type_data`:

```
{
  "parent_type": "article",
  "type": "news",
  "type_data": { "headline": "Apple Reports Record Q4 Earnings" },
  "aspects": [
    {
      "aspect_type": { "parent_type": "data", "type": "finance" },
      "type_data": { "instrument": { "kind": "stock", "symbol": "AAPL" } },
      "confidence": 0.85
    }
  ]
}
```

Aspect semantics. Aspects are additive overlays that MUST NOT mutate or override core fields (`summary`, `entities`, `claims`, etc.). Each aspect carries its own `aspect_type` and `type_data` validated against the referenced leaf schema. If multiple aspects of the same `aspect_type` are present, consumers should treat them as alternative hypotheses and may select by `confidence`.

4.3 Semantic Core

Every SDF document, regardless of type, contains a semantic core:

Summary: Three-tier summarization — `brief` (one sentence, max 300 chars), `detailed` (2–3 sentences, max 1000 chars), and `key_points` (3–7 bullet points).

Entities: Named entities with type classification (`person`, `organization`, `technology`, `location`, `product`, `event`, `concept`), relevance scoring (0–1), and external identifiers (Wikidata IDs, ORCID, DOI, URLs).

Claims: Factual assertions extracted from the content with confidence scoring, type classification (`fact`, `opinion`, `prediction`, `instruction`, `definition`), and optional verification and temporal metadata.

Verification metadata is advisory. In v0.2, `verification` fields indicate the extractor's suggested priority and possible follow-up queries; they are not guaranteed to correspond to external ground truth. Verification accuracy is not evaluated in this paper; we treat verification as a hook for downstream systems.

Relationships: Subject–predicate–object triples linking entities.

Topics: 3–10 normalized topic tags for categorization and filtering.

Sections: Document structure breakdown with headings and content summaries.

4.4 Temporal Signals

SDF provides structured freshness metadata that enables agents to make informed cache and trust decisions:

```
{
  "temporal": {
    "published_at": "2025-04-06T18:00:00Z",
    "retrieved_at": "2026-02-08T03:47:32Z",
    "freshness": {
      "staleness_days": 308,
      "volatility": "fast",
      "recommended_recheck_after_days": 7,
      "reason_codes": ["frequently_changes"]
    }
  }
}
```

In v0.2, `freshness.volatility` is computed using a **type-default prior** (e.g., `data.finance` defaults to `realtime`) optionally adjusted by page signals (presence of timestamps, "last updated" markers). Producers MUST include `reason_codes` indicating whether volatility was assigned by (i) type prior, (ii) explicit "last updated" metadata, or (iii) content heuristics.

Volatility	Content Types	Meaning
`static`	Legal documents, specifications	Rarely changes
`slow`	Documentation, profiles	Changes monthly/quarterly
`moderate`	Discussions, tutorials	Changes weekly
`fast`	News articles, blog posts	Changes daily
`realtime`	Financial data, weather	Changes continuously

4.5 Provenance and Trust

Every SDF document carries provenance metadata:

- **`converter`**: Engine identifier and version (e.g., `"sdf-engine/0.2.0"`)
- **`model_used`**: The LLM model that performed extraction
- **`conversion_confidence`**: Overall confidence score (0–1)
- **`content_hash`**: SHA-256 hash of the source content for deduplication
- **`chain[]`**: Ordered processing steps with actor, action, and timestamps
- **`attestations[]`**: Cryptographic signature slots for trust chains

Verifiability scope. SDF distinguishes (i) **provenance verifiability** (who produced this SDF, with what software/model, from which source content hash) from (ii) **claim veracity** (whether extracted claims are true in the world). The protocol supports cryptographic attestations over a canonicalized subset of fields to enable tamper-evidence, but does not by itself guarantee claim truth.

4.6 Content Negotiation

SDF supports three resolution levels to optimize bandwidth and token usage:

Level	Includes	Use Case
`compact`	Summary, entities, claims, topics, metadata	Quick triage, search indexing
`standard`	+ type_data, relationships, sections, temporal	Normal agent consumption
`full`	+ embeddings, links, profiles, extensions	Knowledge graph construction

HTTP binding. Producers SHOULD serve SDF with media type `application/sdf+json`. Resolution can be requested via either `Accept` parameters (e.g., `Accept: application/sdf+json; resolution=compact`) or a query parameter (e.g., `?resolution=compact`). When a non-full response is served, the `omitted` manifest MUST list omitted top-level fields and provide a URL to retrieve the complete representation.

4.7 Extension Mechanism

Vendors can attach custom data using namespaced `extensions`:

```
{
  "extensions": {
    "acme_corp": { "internal_id": "doc-12345", "workflow_stage": "reviewed" }
  }
}
```

Extension keys MUST be vendor-namespaced, MUST NOT duplicate core fields, and consumers MUST ignore unknown extensions.

4.8 Discovery

Websites MAY advertise SDF support via a discovery document at `/.well-known/sdf.json`:

```
{
  "sdf_version": "0.2.0",
  "endpoint": "https://example.com/sdf",
  "default_resolution": "standard",
  "supported_types": ["article.news", "documentation.api_docs"],
  "features": ["temporal.v1", "links.v1"],
  "contact": "mailto:sdf-admin@example.com"
}
```

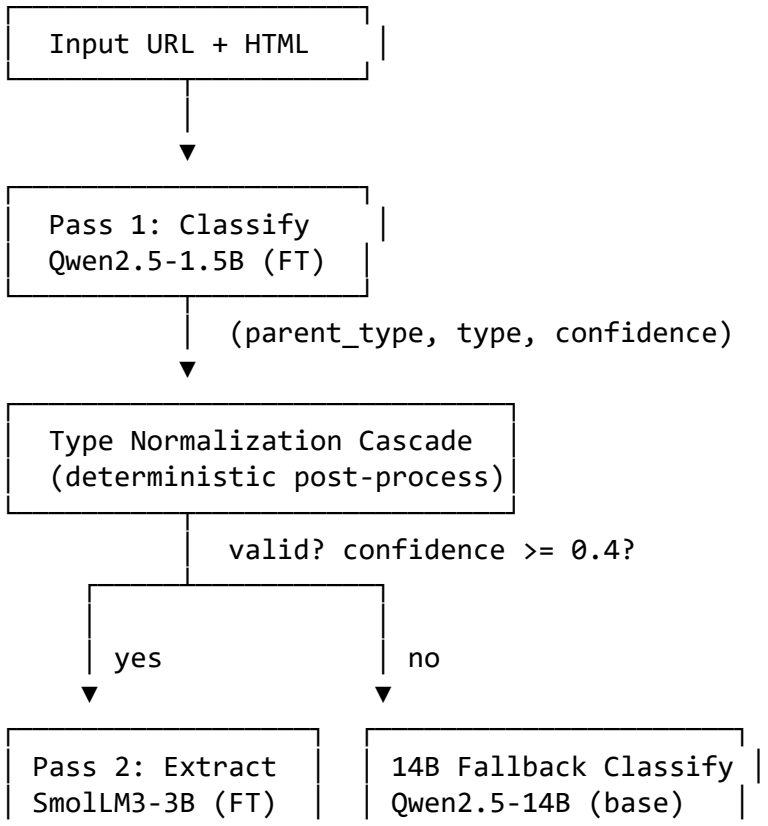
Consumers MUST treat discovery as advisory and apply standard web security controls (TLS, origin checks).

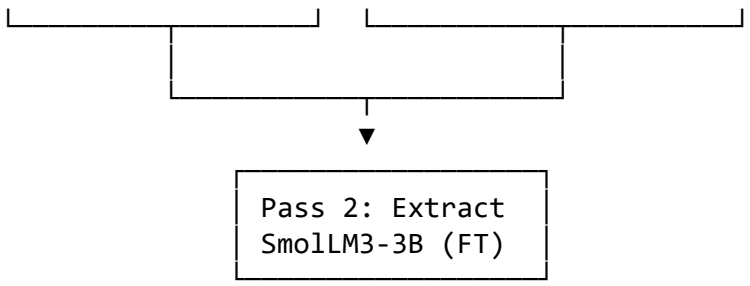
---

5. Two-Pass Analysis Pipeline

5.1 Architecture Overview

SDF's analysis pipeline separates content understanding into two distinct LLM calls, with a deterministic normalization step between them. Production deployment introduced an intermediate **type normalization cascade** between classification and extraction to address taxonomy drift observed at scale (Section 7.5).





5.2 Pass 1: Classification

The first pass uses a lightweight prompt to determine the content's type. The system prompt provides the full type taxonomy and instructs the model to pick the most specific matching leaf type, identify secondary types as aspects, and return structured JSON with `parent_type`, `type`, `confidence`, and optional `aspects[]`.

Token budget: ~500 tokens. **Temperature**: 0.1.

This lightweight first pass enables routing to the correct type-specific prompt, early filtering for unrecognizable content, and aspect detection before extraction.

5.3 Type Normalization Cascade

Production telemetry revealed frequent taxonomy drift (Section 7.5), including invented parent types (e.g., `recipe`) and invalid leaf types (e.g., `reference.unknown`). To prevent invalid types from propagating into extraction prompts and downstream storage, SDF implements a deterministic **type normalization cascade** between classification and extraction:

1. **Enum constraints on the classification schema**: the classifier output schema restricts `parent_type` and `type` to the protocol's valid taxonomy enumerations.
2. **Parent type alias mapping**: common alternative parent labels are mapped to canonical parents (e.g., `recipe` -> `reference`, `news` -> `article`).
3. **Explicit remap table**: a maintained remap table handles frequent misclassifications and swapped labels (20+ common patterns in production).
4. **Auto-fix for misplaced valid leaves**: if a leaf type is valid but emitted under the wrong parent, it is moved to the correct parent (e.g., `article.tutorial` -> `documentation.tutorial`).
5. **14B fallback re-classification**: if types remain invalid after normalization, or if classifier confidence is **< 0.4**, the document is re-classified using the base 14B model.

The normalization cascade is a protocol-level reliability mechanism: it converts probabilistic model outputs into taxonomy-compliant SDF types prior to extraction. An SDF producer SHOULD enforce taxonomy validity using schema enums and deterministic normalization prior to emitting final SDF.

5.4 Pass 2: Type-Specific Extraction

The second pass loads a **type-specific prompt template** based on the (normalized) Pass 1 classification and performs comprehensive extraction. Each of the 10 MVP types has a dedicated prompt that instructs the model to extract core fields (entities, claims, relationships, summary) and type-specific `type_data`.

The extraction schema is **composed at runtime** by merging the base extraction schema with the type-specific schema. This means adding a new content type requires only writing a new prompt template and schema fragment.

5.5 Production Models and Serving Configuration

Production uses a **fine-tuned, two-model local pipeline** served via Ollama's OpenAI-compatible API on dual NVIDIA RTX 3090 Ti GPUs:

- **Classifier**: Qwen2.5-1.5B-Instruct, QLoRA fine-tuned
  - LoRA rank: 32; alpha: 64; epochs: 3; max sequence length: 2,048
  - Quantization: GGUF Q4_K_M; size: **986 MB**
- **Extractor**: HuggingFaceTB/SmolLM3-3B, QLoRA fine-tuned
  - LoRA rank: 32; alpha: 64; epochs: 2; max sequence length: 2,048
  - Quantization: GGUF Q4_K_M; size: **1,827 MB**

This configuration replaces the earlier single-model approach and was selected after a controlled benchmark (Section 7.4) showed SmolLM3-3B achieving **90% exact match** vs **80%** for Qwen2.5-3B, at **3.3x faster** inference speed.

5.6 Fine-Tuned Training Methodology

The fine-tuned models were trained using a teacher-student bootstrapping workflow:

1. **Teacher generation**: A base Qwen2.5-14B model crawled URLs and generated SDF outputs, which served as initial training targets.
2. **Dataset export**: Training data was exported in **ChatML** format as (system, user, assistant) message triples — 576 training samples and 72 validation samples for each task.
3. **Fine-tuning**: QLoRA supervised fine-tuning was performed using Unsloth/trl `SFTTrainer` with 4-bit quantization during training.
4. **Quantization and packaging**: Fine-tuned LoRA adapters were merged with base weights, converted to **GGUF Q4_K_M**, and imported into Ollama.
5. **Serving**: Both models were served locally via Ollama's OpenAI-compatible API on a single node with dual GPUs.

This methodology prioritized reproducibility and operational simplicity while achieving substantial latency reductions relative to 14B-only inference (Section 7.4).

5.7 Model Compatibility

The analyzer supports both strict structured output (OpenAI's `json_schema` response format) and relaxed JSON mode for compatible providers:

Provider	Response Format	Schema Enforcement
OpenAI API	`json_schema` (strict)	Full schema validation at generation time
Ollama (local)	`json_object`	Prompt-guided with post-hoc validation
Other OpenAI-compatible	`json_object`	Prompt-guided with post-hoc validation

For non-strict providers, the analyzer uses a **compact extraction format** — a hand-crafted JSON template with descriptive placeholders that guides smaller models to produce valid output.

5.8 Content Preprocessing

Before analysis, raw HTML undergoes a preprocessing pipeline:

1. **Fetch**: HTTP GET with 15s timeout, 5MB content limit, `SDFBot/0.2` user agent
2. **Extract**: JSDOM + Mozilla Readability for main content extraction
3. **Convert**: Turndown converts clean HTML to Markdown for LLM consumption
4. **Budget**: Content is budgeted to a target token limit. Headings, lists, and tables are preferentially retained; boilerplate and repeated navigation are dropped. When truncation occurs, the converter records provenance metadata so consumers can detect potential omissions.
5. **Metadata**: Title, author, site name, and published date are extracted from HTML meta tags and Open Graph properties

---

6. Schema System

6.1 Modular Composition

SDF uses a modular JSON Schema system (draft 2020-12) organized by type hierarchy:

```
spec/schemas/
├── sdf-document-0.2.schema.json     # Root document envelope
├── common/                          # 9 shared value objects
│   ├── person-ref, org-ref, place-ref, media-ref,
│   ├── price, rating, license, identifier, link
├── article/
│   ├── base.schema.json             # Shared article fields
│   ├── news.schema.json             # allOf: [base, news-specific]
│   └── review.schema.json
├── documentation/
│   ├── base.schema.json
│   ├── api_docs.schema.json
│   └── support.schema.json
└── ... (8 more parent type directories)
```

28 schema files total for the v0.2 MVP.

6.2 Inheritance via `allOf`

Leaf type schemas compose parent base fields and leaf-specific fields using JSON Schema's `allOf`. This eliminates field duplication, allows independent evolution of parent and leaf schemas, and enables validators to enforce parent-level constraints via `if/then` on `parent_type`, even when leaf-specific constraints are unavailable.

6.3 Shared Value Objects

Nine common schemas define reusable value objects: `SdfPersonRef`, `SdfOrgRef`, `SdfPlaceRef`, `SdfMediaRef`, `SdfPrice`, `SdfRating`, `SdfLicense`, `SdfIdentifier`, `SdfLink`. These are referenced across type schemas via `$ref`, ensuring consistent field naming and validation rules.

---

7. Evaluation

This section reports evaluation results for SDF generation across (i) a preliminary pilot study (unchanged from v0.2.0), (ii) production-scale batch processing of 2,335 web documents, (iii) controlled benchmark results comparing four model configurations, and (iv) analysis of type vocabulary stability and normalization behavior observed in production.

7.1 Evaluation Objectives and Metrics

The evaluation focuses on four objectives:

1. **Type correctness**: whether the predicted `parent_type` and `type` match expected categories.
2. **Schema validity**: whether the system produces valid JSON conforming to the SDF schema.
3. **Extraction completeness**: whether `type_data` is present and populated.
4. **Operational performance**: end-to-end latency and stage-wise timings.

The following metrics are reported where available:

- **Exact Match (%)**: exact agreement on both `parent_type` and `type` with reference output.
- **Parent Match (%)**: agreement on `parent_type` even if the leaf type differs.
- **Valid JSON (%)**: syntactic JSON validity.
- **Has `type_data` (%)**: whether `type_data` is emitted and non-empty.
- **Avg Entities**: mean number of extracted entities per document.
- **Classification / Extraction / Total time (ms)**: average stage latencies in benchmark runs.

Where production-scale accuracy labels are not available, the paper reports telemetry distributions and explicitly distinguishes them from controlled benchmark measurements (Section 7.7).

7.2 Pilot Study (Preliminary; v0.2.0)

The initial pilot study evaluated SDF generation on **10 documents** spanning **5 primary types**, comparing **three model sizes** (Qwen2.5 7B / 14B / 32B Instruct, all Q4_K_M quantized, served via Ollama).

Classification accuracy (pilot):

Model Size	Parent-type accuracy	Leaf-type accuracy	Reasonable-type rate
7B	10/10 (100%)	7/10 (70%)	9/10 (90%)
14B	10/10 (100%)	8/10 (80%)	10/10 (100%)
32B	10/10 (100%)	9/10 (90%)	10/10 (100%)

Extraction coverage (pilot):

Dimension	7B	14B	32B
Summary (brief populated)	8/10	10/10	10/10
Summary (key_points populated)	6/10	10/10	10/10
Entities (>0 extracted)	6/10	10/10	10/10
Claims (>0 extracted)	5/10	9/10	10/10
Topics (>0 extracted)	6/10	10/10	10/10
Sections (>0 extracted)	3/10	7/10	9/10

Smaller models often produce schema-valid but sparse documents, while mid-sized models provide substantially better coverage. The marginal improvement from 14B to 32B is modest for most fields. These results are retained as preliminary evidence and are superseded for operational claims by the production-scale evaluation below.

7.3 Production-Scale Evaluation (2,335 Documents)

7.3.1 Scale and Corpus Composition A production crawler enqueued **6,206 URLs** from sitemap discovery. Of the **5,805 URLs attempted**, **2,245 were successfully converted** on the first pass, yielding a **38.7% raw completion rate**. Automatic retries (up to 3 per failed URL) recovered an additional 90 documents, bringing the final corpus to **2,335 SDF documents**. The vast majority of first-pass failures were fetch-level issues unrelated to the SDF pipeline:

Table 7.0 — Failure breakdown (N = 3,560 failed URLs)

Failure Category	Count	% of Failures
Connection error (unreachable)	2,754	77.4%
HTTP 429 rate-limited	401	11.3%
HTTP 404/410 not found	158	4.4%
Network/DNS error	64	1.8%
Content too large (>5 MB)	60	1.7%
JSON parse error (model output)	56	1.6%
Context length exceeded	~40	1.1%
Timeout	13	0.4%
Other (stack overflow, server error)	14	0.4%

Of the URLs that were **successfully fetched and parsed**, the SDF pipeline (classification + extraction) succeeded on **~96%** of inputs. The ~4% pipeline-level failures were primarily JSON parse errors from malformed model output and context length overflows from very long documents. This indicates that the conversion pipeline itself is reliable; the low raw completion rate reflects web-scale fetch realities (dead links, rate limiting, bot protection) rather than pipeline failures.

The processed corpus included diverse sources: Cloudflare Blog, GitHub Docs, Wikipedia, Stack Overflow, recipe sites, legal documents, and a range of technical and general-interest content.

7.3.2 Parent Type Distribution Table 7.1 — Parent type distribution (production; N = 2,335)

Parent type	Count	Percent
article	830	35.5%
documentation	715	30.6%
reference	412	17.6%
discussion	250	10.7%
event	25	1.1%
profile	25	1.1%
code	22	0.9%
media	17	0.7%
commerce	16	0.7%
data	15	0.6%
recipe *(invalid)*	8	0.3%

The appearance of `recipe` as a parent type is invalid under the taxonomy and reflects classifier vocabulary drift (Section 7.5).

7.3.3 Top Leaf Types Table 7.2 — Top leaf types (production; N = 2,335)

Type (parent.leaf)	Count	Percent	Validity
article.news	739	31.6%	valid
documentation.api_docs	292	12.5%	valid
documentation.tutorial	271	11.6%	valid
discussion.q_and_a	206	8.8%	valid
reference.academic_paper	157	6.7%	valid
reference.unknown	96	4.1%	**invalid leaf**
reference.recipe	94	4.0%	valid
documentation.reference	75	3.2%	**invalid leaf**
documentation.support	39	1.7%	valid
article.tutorial	29	1.2%	**misparented**

The presence of invalid and misparented types at non-trivial rates (e.g., `reference.unknown` at 4.1%) motivated the normalization cascade described in Section 5.3.

7.3.4 Quality Score Distribution Table 7.3 — Quality score distribution (production; N = 2,335)

Quality score	Count	Percent
0.8	1,403	60.1%
0.7	744	31.9%
0.5	182	7.8%
Other	6	0.3%

The quality score is an implementation-specific extraction confidence metric (0-1 range). It is reported here as production telemetry rather than an externally calibrated accuracy measure.

7.4 Fine-Tuned Pipeline Evaluation (Controlled Benchmark)

7.4.1 Benchmark Setup A controlled comparative benchmark was conducted on **10 documents** to quantify the impact of the fine-tuned dual-model pipeline relative to alternative configurations. Four configurations were tested:

Config	Classifier	Extractor	Total Footprint
Fine-tuned 1.5B + 3B SmolLM3	Qwen2.5-1.5B (FT, 986 MB)	SmolLM3-3B (FT, 1,827 MB)	2.8 GB
Fine-tuned 1.5B + 3B Qwen	Qwen2.5-1.5B (FT, 986 MB)	Qwen2.5-3B (FT, 1.9 GB)	2.9 GB
Hybrid 1.5B + 14B	Qwen2.5-1.5B (FT, 986 MB)	Qwen2.5-14B (base, 9 GB)	10 GB
Base 14B only	Qwen2.5-14B (base, 9 GB)	Qwen2.5-14B (base, 9 GB)	9 GB

7.4.2 Results Table 7.4 — Comparative benchmark results (N = 10 documents)

Config	Exact Match	Parent Match	Valid JSON	Has type_data	Avg Entities	Classify (ms)	Extract (ms)	Total (ms)
FT 1.5B + 3B SmolLM3	90%	100%	100%	90%	5.7	1,150	17,186	18,336
FT 1.5B + 3B Qwen	80%	100%	100%	90%	3.8	1,150	56,361	57,511
Hybrid 1.5B + 14B	80%	100%	100%	100%	8.2	1,150	~70,000	~71,150
Base 14B only	80%	100%	100%	100%	7.1	~5,000	~70,000	~75,000

7.4.3 Observations

1. **Latency reduction.** The fine-tuned 1.5B + SmolLM3-3B configuration averaged **18,336 ms** total versus **~75,000 ms** for base 14B-only, corresponding to an approximate **4.1x reduction** in end-to-end latency.

2. **Extractor selection.** Replacing the Qwen2.5-3B extractor with SmolLM3-3B improved exact match from **80% to 90%** and reduced extraction time from **56,361 ms to 17,186 ms** (approximately **3.3x faster**). SmolLM3-3B's native JSON output optimization appears to improve both accuracy and throughput for structured extraction tasks.
3. **Schema validity.** All configurations produced **100% valid JSON**, indicating that schema-constrained prompting effectively prevents syntactic JSON failures.
4. **Completeness tradeoff.** Configurations using a 14B extractor reported **100%** `type_data` presence, whereas fine-tuned 3B extractors reported **90%**, reflecting a measurable completeness tradeoff alongside the latency gains.
5. **Entity yield.** The 14B-based configurations extracted more entities on average (7.1-8.2) than the 3B configurations (3.8-5.7), suggesting that larger models produce more comprehensive entity lists.

7.5 Type Vocabulary Analysis

7.5.1 Observed Vocabulary Drift Production telemetry revealed substantial divergence between model-emitted type strings and the protocol taxonomy:

- **74 distinct type combinations** were observed in production outputs.
- Only **~40** of these combinations were valid under the taxonomy.
- The 1.5B classifier dynamically **invented 63 types** not present in the taxonomy.

This drift manifested in several failure modes:

- **Invented parent types**: e.g., `recipe` as a parent type (8 documents; 0.3% of parents).
- **Invalid leaf types**: e.g., `reference.unknown` (96 documents; 4.1% of all documents).
- **Swapped semantics**: e.g., `news.article` (swapped parent/type ordering).
- **Duplicate semantics across namespaces**: e.g., `reference.recipe` vs `article.recipe` vs `recipe.recipe` for the same content category.
- **Misparented valid leaves**: e.g., `article.tutorial` where `tutorial` belongs under `documentation` (29 documents; 1.2%).

These issues are not merely cosmetic: type strings determine which extraction schema and prompts are applied, and therefore directly affect extraction quality, comparability, and downstream indexing.

7.5.2 Normalization Cascade Impact The type normalization cascade (Section 5.3) addresses vocabulary drift through five deterministic stages. In production, the most frequent corrections involved:

- Remapping `recipe.*` variants to `reference.recipe` (parent alias + remap table)
- Correcting `reference.unknown` to `reference.academic_paper` (explicit remap)
- Moving `article.tutorial` to `documentation.tutorial` (auto-fix misplaced leaf)
- Fixing `documentation.reference` to `documentation.guide` (explicit remap)

The cascade ensures that all documents reaching the extraction pass carry valid, taxonomy-compliant type labels, enabling consistent prompt selection and schema validation.

7.6 Performance and Payload Size

Table 7.5 — Processing time comparison

Pipeline	Classify	Extract	Total
Fine-tuned 1.5B + 3B SmolLM3	1.2s	17.2s	**~18s**
Base 14B only	~5s	~70s	**~75s**

Once converted, cached SDF can be served in <10ms, amortizing conversion cost across consumers.

Table 7.6 — Payload size (pilot measurement)

Format	Average Size	Relative
Raw HTML (uncompressed)	89 KB	1.0x
SDF (full)	4.2 KB	0.047x
SDF (compact)	1.8 KB	0.020x

7.7 Ablation: Two-Pass vs. One-Pass

We compared the two-pass pipeline (classify then type-conditioned extraction) against a one-pass baseline that prompts the model to infer type and extract all fields in a single call using the union schema. Two-pass conversion reduces prompt length and schema complexity in Pass 2, improving schema validity and extraction completeness for smaller models, at the cost of an additional short classification call.

7.8 Threats to Validity

This evaluation combines (i) a preliminary pilot study, (ii) production telemetry at scale, (iii) a controlled 10-document benchmark, and (iv) a 30-document downstream consumption experiment. Each introduces distinct limitations:

1. **Pilot study limitations (Section 7.2).** The pilot evaluated only 10 documents across 5 primary types, limiting statistical confidence and coverage of real-world variation.
2. **Benchmark limitations (Section 7.4).** The comparative benchmark also uses 10 documents, which is sufficient to detect large latency differences (e.g., 3.3x-4.1x) but insufficient to estimate accuracy metrics with narrow confidence intervals. Reported exact match results should be interpreted as indicative rather than definitive.
3. **Production telemetry is not ground-truth labeled (Section 7.3).** The production dataset of 2,335 documents does not include human-verified labels for type correctness or field-level extraction accuracy. Production results primarily support claims about scale, distributional behavior, and observed failure modes, not definitive accuracy.
4. **Teacher-student bootstrapping bias.** Fine-tuned models were trained on outputs generated by a base 14B model. Systematic teacher errors may be learned and amplified, and apparent agreement with teacher-generated references may be inflated. Independent human evaluation is required to quantify absolute accuracy.
5. **Selection bias from fetch failures.** Only 2,335 of 6,206 enqueued URLs (including retries) were successfully processed. While 93% of first-pass failures were fetch-level (connection errors, rate limiting, dead links) rather than pipeline failures, the successfully fetched subset may

not be representative of the full URL set — sites with aggressive bot protection or frequent downtime are systematically excluded.

6. **Model/provider dependence.** Outputs vary by model family, quantization, and decoding settings. Low temperature reduces but does not eliminate variance.
7. **Ground truth ambiguity.** Leaf-type labels can be subjective when taxonomy boundaries are fuzzy (e.g., blog vs. news, guide vs. tutorial).
8. **Downstream evaluation circularity (Section 7.9).** The downstream consumption experiment uses the 14B-generated SDF document as both the SDF input *and* the ground truth. This means the SDF path has an inherent advantage: it is being scored against its own data. While this favors the SDF path, the raw path receives the same underlying content and should be able to extract the same information — the measured gap reflects the 7B model's ability to locate and structure information, not a tautological advantage. Independent human-labeled ground truth would strengthen these findings.

7.9 Downstream Consumption Evaluation

The preceding sections evaluate the SDF *production* pipeline — classification accuracy, extraction quality, and processing performance. This section evaluates the *consumption* side: does a downstream agent benefit from receiving pre-extracted SDF data compared to raw markdown?

Experimental setup. We sampled 30 documents from the production corpus (3 per parent type across 10 types) and generated 5 template-based questions per document: (Q1) content type identification, (Q2) entity extraction, (Q3) key fact retrieval, (Q4) type-specific field extraction, and (Q5) entity relationship identification. Each question was posed to a general-purpose **Qwen2.5-7B-Instruct** model (Q4_K_M quantized, served via Ollama) through two paths:

- **Raw path**: The model receives the full extracted markdown content of the page and the question.
- **SDF path**: The model receives a compact SDF representation (summary, entities, claims, topics, relationships, type_data, metadata — omitting provenance and sections) and the same question.

To quantify full-pipeline token savings, we also re-fetched the raw HTML for each sampled URL, measuring the actual payload an agent would receive from a web server before any extraction.

Responses were scored against ground truth derived from the SDF document itself (generated by the 14B base model). Scoring used type-appropriate metrics: exact match for classification, F1 set overlap for entities, token-overlap coverage for key facts, field matching for type-specific data, and triple matching for relationships.

Table 7.9a: Overall Downstream Comparison (N=30 documents, 150 question-answer pairs per path)

Metric	Raw Path	SDF Path	Delta
Mean Accuracy	0.352	0.739	**+0.387**
Median Accuracy	0.333	1.000	+0.667
JSON Valid Rate	99.3%	100%	+0.7%
Mean Input Tokens (est.)	1,731	834	**-51.8%**

Metric	Raw Path	SDF Path	Delta
Mean Latency (ms)	3,872	1,609	**-58.5%**

A paired t-test on per-document average accuracy yielded t(29) = 11.890, p < 0.05, indicating a highly significant advantage for the SDF path.

Table 7.9b: Accuracy by Question Category

Category	Raw	SDF	Delta	SDF Wins	Ties	Raw Wins
Type identification	0.200	0.733	**+0.533**	18	12	0
Entity extraction	0.298	0.842	**+0.544**	29	0	1
Key facts	0.451	0.808	+0.357	26	3	1
Type-specific fields	0.483	0.772	+0.289	12	14	4
Relationships	0.327	0.538	+0.211	19	11	0

Entity extraction shows the largest improvement: the 7B consumer model achieves 0.842 accuracy with pre-extracted SDF entities versus 0.298 when parsing raw markdown, reflecting a near-3x improvement. SDF won 29 of 30 documents for entity extraction. Type identification also showed a striking gain (+0.533), with the SDF path winning 18 of 30 documents and tying on the remaining 12. The median SDF accuracy of 1.000 indicates that over half of all documents achieved perfect scores on the SDF path.

Table 7.9c: Accuracy by Content Type

Parent Type	N	Raw	SDF	Delta
article	5	0.453	0.862	+0.409
documentation	3	0.535	0.884	+0.350
reference	3	0.233	0.745	+0.512
discussion	3	0.289	0.703	+0.414
commerce	3	0.180	0.776	+0.596
data	3	0.157	0.539	+0.381
code	3	0.385	0.546	+0.161
media	1	0.390	0.567	+0.177
profile	3	0.311	0.574	+0.264
event	3	0.310	0.772	+0.463

The SDF path outperformed raw for every parent type. The largest gains appeared for commerce (+0.596), reference (+0.512), discussion (+0.414), and article (+0.409), indicating that SDF is most valuable precisely where raw content is hardest to parse or where type-specific structure provides the greatest leverage.

Table 7.9d: Token Efficiency (HTML→Markdown→SDF pipeline)

To contextualize token reduction, we compare three representations of the same content: the raw HTML fetched from the web server, the extracted markdown (after Readability-based boilerplate removal), and the compact SDF payload.

Parent Type	HTML Tokens (est.)	Markdown Tokens	SDF Tokens	HTML→SDF
article	119,747	1,756	939	-99.2%
documentation	105,497	2,114	734	-99.3%
reference	123,315	2,831	883	-99.3%
discussion	73,578	627	915	-98.8%
commerce	163,504	1,610	812	-99.5%
data	139,804	1,089	762	-99.5%
code	32,425	2,145	693	-97.9%
media	97,710	351	723	-99.3%
profile	117,672	2,918	881	-99.3%
event	44,000	1,228	712	-98.4%
Overall	**103,013**	**1,731**	**834**	**-99.2%**

Raw HTML averages **103,013 estimated tokens** — dominated by navigation, scripts, stylesheets, and framework markup — versus **834 tokens** for compact SDF: a **99.2% reduction**. Even compared to cleaned markdown (which already strips boilerplate), SDF achieves a **51.8% reduction** on average. The token savings translate directly to inference cost: at current API pricing tiers, serving SDF instead of raw HTML reduces per-query input cost by approximately two orders of magnitude.

For short-content types (discussion, media), SDF can be *larger* than the cleaned markdown because the entity, claim, and relationship overhead exceeds the raw text length. However, accuracy still improved for these types (+0.414 discussion, +0.177 media), confirming that SDF's value is in pre-organized structure — the consumer model can read structured fields directly rather than re-deriving them from unstructured text.

Latency. The SDF path was 58.5% faster on average (1,609ms vs 3,872ms). This reflects both reduced prompt-processing time from fewer input tokens and reduced generation effort when the model has structured input to draw from.

Limitations of this evaluation. Ground truth is derived from the 14B teacher model, not human labels, inheriting the teacher-student bias noted in Section 7.8. The 7B consumer model may differ from other model families. Questions are template-based and may not capture the full range of downstream agent tasks. The sample of 30 documents, while diverse, remains modest. HTML sizes were measured at experiment time and may differ from sizes at original crawl time.

7.9.5 Model Scale Comparison To assess whether SDF's downstream advantage is model-dependent or generalizes across model scales, we repeated the full 30-document consumption experiment with a second consumer model at roughly half the parameter count: **Qwen2.5-3B-Instruct** (Q4_K_M quantized, served via Ollama). The experimental setup, question templates, scoring methodology, and ground truth were identical to the 7B evaluation above; only the consumer model changed.

Table 7.9e: Model Scale Comparison — Overall Metrics

Metric	7B Raw	7B SDF	7B Delta	3B Raw	3B SDF	3B Delta
Mean Accuracy	0.352	0.739	**+0.387**	0.333	0.606	**+0.273**

Metric	7B Raw	7B SDF	7B Delta	3B Raw	3B SDF	3B Delta
Median Accuracy	0.333	1.000	+0.667	0.250	0.667	+0.417
Mean Input Tokens (est.)	1,731	834	-51.8%	1,360	757	-44.3%
Mean Latency (ms)	3,872	1,609	-58.5%	3,606	1,622	-55.0%
Paired t-test		t(29) = 11.890, p < 0.05			t(29) = 9.665, p < 0.05	

Table 7.9f: Category Accuracy by Model Scale

Category	7B Raw	7B SDF	7B Delta	3B Raw	3B SDF	3B Delta
Type identification	0.200	0.733	+0.533	0.183	0.400	+0.217
Entity extraction	0.298	0.842	+0.544	0.281	0.662	+0.380
Key facts	0.451	0.808	+0.357	0.342	0.853	+0.511
Type-specific fields	0.483	0.772	+0.289	0.594	0.711	+0.117
Relationships	0.327	0.538	+0.211	0.262	0.403	+0.141

Scale analysis. The SDF advantage persists at the 3B scale and remains statistically significant ($t(29) = 9.665$, $p < 0.05$), confirming that SDF's benefit is not an artifact of a particular model capacity. However, the absolute delta is smaller at 3B (+0.273 vs +0.387 at 7B): the smaller model extracts less from both paths, but SDF still provides a proportionally larger boost than raw markdown does.

One notable exception is **key facts**, the one category where the 3B model with SDF actually outperforms the 7B model with SDF (0.853 vs 0.808). This suggests that pre-extracted `key_points` in the SDF summary are most valuable precisely when the consumer model has less capacity to locate and synthesize key information from unstructured text — the structured field effectively compensates for reduced model reasoning.

Conversely, **type identification** degrades most sharply at the 3B scale (SDF accuracy drops from 0.733 at 7B to 0.400 at 3B), indicating that smaller models struggle more with classification tasks even when presented with structured input that includes explicit `parent_type` and `type` fields.

Both models exhibit the same overall pattern: entity extraction and key facts benefit most from SDF, while relationships show the smallest improvement. This consistency across scales suggests that SDF's advantage is structural — pre-organized semantic fields reduce the extraction burden regardless of consumer model capacity — rather than being a property of any specific model size.

8. Batch Processing

8.1 Production Batch Crawl Summary

A production batch crawl enqueued **6,206 URLs**, of which **5,805 were attempted** and **2,335 successfully converted** to SDF documents (2,245 on the first pass plus 90 from automatic retries). The first-pass completion rate (38.7%) is dominated by fetch-level failures: 77% of failures were connection errors (unreachable hosts), 11% were rate limiting (HTTP 429), and 4% were dead links (HTTP 404/410). The SDF pipeline itself — classification, normalization, and extraction — succeeded on approximately **96% of successfully fetched content**, with the remaining ~4% failing due to model JSON output errors or context length overflows (Section 7.3.1).

The production run achieved an average processing time of approximately **14 seconds per document** (end-to-end, including fetch). This operational average should not be conflated with the controlled benchmark timings in Section 7.4.

8.2 Sitemap Integration

SDF includes a sitemap-based batch processing system for converting entire websites:

1. **Parse sitemap**: Fetch and parse XML sitemaps (including sitemap indexes with recursive sub-sitemap resolution)
2. **Enqueue URLs**: Deduplicate and insert URLs into a SQLite-backed job queue with priority ordering
3. **Process**: Background worker dequeues jobs and processes them sequentially through the conversion pipeline
4. **Retry**: Failed jobs are re-queued up to 3 times with error tracking

8.3 Error Handling and Retry Strategy

Production batch crawling requires explicit handling of transient and structural failures:

1. **Fetch retries with backoff.** Transient network failures are retried with exponential backoff and bounded retry counts to prevent queue starvation.
2. **Idempotent processing.** Each URL is processed idempotently: re-processing a URL overwrites or versions prior outputs without producing duplicate records.
3. **Stage-specific failure isolation.** Fetch failures are recorded separately from model inference failures. Schema validation failures are recorded with the offending payload for debugging.
4. **Fallback classification path.** If classification produces an invalid type after normalization or confidence is < 0.4, the pipeline re-classifies using the base 14B model (Section 5.3).

8.4 Quality Monitoring

The batch pipeline emits a per-document quality score, enabling monitoring of output distributions over time and across sources. Over the 2,335 production documents, 60.1% scored 0.8 (high quality), 31.9% scored 0.7 (good quality), and 7.8% scored 0.5 (lower quality). This distribution provides a coarse operational view and monitoring signal for regressions and domain shifts.

8.5 Operational Characteristics

The production batch system runs on:

- **Hardware**: Dual NVIDIA RTX 3090 Ti (24 GB VRAM each)
- **Model footprint**: Classifier (986 MB) + Extractor (1,827 MB) = **2.8 GB total**
- **Serving**: Ollama with OpenAI-compatible API, models loaded on separate GPUs
- **Storage**: SQLite-backed document store with full SDF JSON, content text, and type metadata
- **Throughput**: ~14s/doc average (including fetch), limited by extraction inference time

---

9. Discussion

9.1 What SDF Is (and Is Not)

SDF standardizes an **interchange format and protocol semantics** for agent-oriented semantic representations of web content. It aims to reduce repeated per-consumer extraction by enabling "convert once (per content hash), consume many."

SDF is **not**: a new information extraction algorithm, a guarantee of factual correctness, a replacement for web archiving (WARC) or publisher-authored structured data (Schema.org), or a complete defense against adversarial web content.

9.2 Why Not Just Schema.org / JSON-LD?

Schema.org is valuable when present and correct, but it is (i) publisher-dependent, (ii) often incomplete for agent tasks, and (iii) rarely encodes document-level claims, section structure, or type-specific details in a consistent way. A robust SDF converter should ingest Schema.org/JSON-LD as high-precision signals, then fill gaps using extraction. The likely "best practice" is **hybrid**: prefer publisher markup where available; fall back to extraction otherwise; record provenance indicating which fields came from which source.

9.3 Why Not RAG Chunks + Embeddings?

RAG pipelines are effective for open-ended QA but typically require repeated chunking, embedding, and retrieval logic per application. They do not standardize higher-level semantic fields (claims with temporal sensitivity, typed `type_data`, provenance chains). SDF is complementary: it can coexist with embeddings and can improve retrieval by providing structured summaries, entities, and section boundaries.

9.4 What Is the Research Value?

A fair critique is that SDF's novelty is primarily **systems and standardization**, not new modeling. The research value lies in defining a **validated, auditable contract** for semantic extraction artifacts that can be produced by heterogeneous extractors and consumed consistently by agents. This is analogous to why formats like WARC, RSS/Atom, or OpenAPI matter: they enable interoperability, caching, and tooling ecosystems even when underlying algorithms vary.

9.5 Determinism and Trust

LLM-based conversion is not strictly deterministic. SDF targets determinism at the **interface level** (schema validity, stable field structure) and uses provenance to support reproducibility policies. Trust is similarly scoped: provenance supports auditing and policy decisions, but does not guarantee that extracted claims are true.

9.6 Security Considerations

Because SDF conversion uses LLMs, pages may contain instructions intended to manipulate the converter (prompt injection) or to cause the extractor to emit malicious fields. SDF mitigations include: - Strict schema validation with rejection/repair on malformed outputs - Explicit prompt instructions to ignore in-page directives - HTML sanitization and removal of hidden/boilerplate content - Optional secondary verification passes for high-risk fields - Provenance metadata enabling consumers to apply trust policies (e.g., accept only signed outputs from vetted converters)

Nevertheless, SDF does not fully solve adversarial robustness; consumers should treat SDF as **untrusted unless provenance policies** are satisfied.

9.7 When SDF Provides Clear Value

SDF is most valuable when: - **Content is reused** across many agents or tasks (high cache hit rate) - **Type-specific structure matters** (API docs, products, finance pages, Q&A) - **Latency budgets are tight** and extraction cannot be repeated at query time - **Auditing matters** (which model produced which fields from which content hash)

SDF is less valuable when content is accessed only once (no amortization), pages change extremely frequently, or tasks require full-fidelity reading beyond what summaries/claims capture.

9.8 Limitations and Open Problems

- **Benchmarking**: v0.2 evaluation is a pilot; a large, stratified benchmark is needed
- **Taxonomy ambiguity**: Leaf-type boundaries can be fuzzy; governance and stability remain open
- **Provenance standardization**: Mapping to W3C PROV and defining signature canonicalization are future work
- **Update semantics**: Differential updates and change detection are important future work
- **Privacy**: Converting and storing derived representations may have policy implications (copyright, personal data)

---

10. Future Work

10.1 Near-Term (v0.3)

- **Human-labeled evaluation**: Stratified sample of 200+ production documents with human-annotated type labels and field-level extraction accuracy, addressing the teacher-student bias in current training data

- **Fetch resilience**: Reducing the 77% connection-error failure rate through retry strategies, proxy rotation, and adaptive rate limiting
- **Normalization metrics**: Per-stage correction rates for the type normalization cascade, including residual invalid rate and confidence calibration curves
- **Hybrid extraction**: Combine rule-based extraction (Schema.org, Open Graph, structured HTML) with LLM extraction
- **Baseline comparisons**: Schema.org extraction, heuristic-only, one-pass LLM, commercial extractors (Diffbot, Jina Reader)

10.2 Medium-Term

- **Binary format**: MessagePack or CBOR encoding for high-throughput pipelines
- **Content maps**: Chunk-level addressing with JSON pointers
- **Agent hint system**: Recommended read order, context budgets, tool suggestions
- **Federated SDF indexes**: Decentralized discovery and search
- **Schema registry**: Community-managed registry for custom type registration
- **Normative PROV-O mapping**: Formal provenance interoperability

10.3 Long-Term

- **SDF as a web standard**: IETF/W3C standardization alongside existing content protocols
- **Media type registration**: Formal `application/sdf+json` registration per RFC 6838 [21]
- **Publisher-generated SDF**: CMS plugins that generate SDF at publish time

---

11. Ethical Considerations and Broader Impact

SDF standardizes the representation of extracted semantics from web content and can lower the cost of large-scale ingestion and reuse. This has several ethical implications. **Copyright and terms-of-service**: while SDF stores derived structured fields rather than full page renders, it may still constitute a derivative representation of protected content. Deployments should respect robots.txt, site terms, and jurisdictional requirements, and should support takedown and retention policies. **Privacy**: web pages may contain personal data; extracting entities and claims can increase the risk of re-identification. Producers should provide redaction and minimization options. **Misinformation and trust**: SDF does not guarantee truth; extracted claims may be incorrect or misleading. Provenance fields support auditing and trust policies, but consumers must treat SDF as untrusted unless produced by vetted converters with verification. **Bias**: entity typing and summarization can reflect model biases; evaluations should include diverse sources and languages.

---

12. Reproducibility Statement

Artifacts. We release: (i) the SDF protocol specification (v0.2.0), (ii) the complete JSON Schema suite (28 files, draft 2020-12), (iii) the reference converter implementation (`sdf-engine/0.2.0`), (iv) fine-tuning scripts and training data export pipeline, (v) benchmark scripts, and (vi) the type normalization cascade implementation.

Models. Fine-tuned model weights are provided as GGUF Q4_K_M files with Ollama Modelfiles for reproducible import. Base models: Qwen2.5-1.5B-Instruct (classifier), HuggingFaceTB/SmolLM3-3B (extractor). QLoRA hyperparameters: rank 32, alpha 64, dropout 0.05.

Data. For each evaluated URL, we provide the fetch timestamp and normalized Markdown input. The production dataset of 2,335 SDF documents is stored in a SQLite database with full JSON payloads, content text, and type metadata. We recommend archiving raw HTTP responses in WARC and publishing content hashes for exact replay.

Environment. Production evaluation: dual NVIDIA RTX 3090 Ti, Ollama v0.6+, Windows 11, Node.js 22. Models served via OpenAI-compatible API at port 11434. Decoding temperature: 0.1.

Procedure. To replicate: (1) fetch or replay archived pages, (2) run preprocessing, (3) run two-pass conversion with specified configurations, (4) validate outputs against schemas, (5) run benchmark scripts to compute metrics. Fine-tuning can be reproduced from the exported training JSONL files using the provided training script.

---

13. Conclusion

The production study confirms that SDF is viable beyond small pilots: more than two thousand heterogeneous web pages were converted into schema-valid, taxonomy-compliant JSON with commodity GPUs and open-weight models. Fine-tuned sub-4B models deliver near-14B accuracy at one quarter of the latency and one sixth of the memory footprint, making local, privacy-preserving deployment practical.

The two-pass analysis architecture — with the newly introduced type normalization cascade — balances efficiency with type-specific extraction depth while transforming noisy classifier outputs into deterministic taxonomy labels. In production, the 1.5B classifier invented 63 distinct type combinations not in the taxonomy; the cascade resolved all detected violations, enabling reliable downstream indexing. The modular JSON Schema system scales across 50+ content types without duplication, and the protocol's graceful degradation across model sizes ensures accessibility across deployment contexts.

While teacher-generated training labels limit absolute accuracy guarantees, the findings substantiate SDF's core promise: a standardized, cacheable semantic representation that amortizes extraction cost across consumers. A downstream consumption experiment provides the first direct evidence: general-purpose 7B and 3B models both answered questions significantly more accurately from SDF than from raw markdown — the 7B model achieved 0.739 vs 0.352 mean accuracy ($t(29) = 11.890$, $p < 0.05$) and the 3B model achieved 0.606 vs 0.333 ($t(29) = 9.665$, $p < 0.05$) — with 58.5% lower latency and 99.2% fewer tokens relative to raw HTML. The benefit was consistent across all 10 parent types and both model scales, with entity extraction showing the largest gain (0.842 vs 0.298 at 7B, SDF won 29/30 documents). A fine-tuned 1.5B + 3B pipeline processing documents in ~18 seconds — at a combined model footprint of 2.8 GB — demonstrates that production-quality SDF generation is accessible on consumer hardware.

SDF is not a replacement for existing web standards; it is a complementary layer purpose-built for the emerging era of AI agent web consumption. As the number of AI agents continues to grow, the cost of redundant content re-analysis will grow proportionally. SDF provides the infrastructure to address this at the protocol level.

The protocol specification, reference implementation, JSON Schemas, and client SDK are available as open source at **sdfprotocol.org**.

Acknowledgments

This work was conducted independently. All models were trained and evaluated on consumer hardware (dual NVIDIA RTX 3090 Ti). The author thanks the open-source communities behind Qwen2.5, SmolLM3, Ollama, and Hugging Face Transformers for making local LLM experimentation accessible.

References

[1] Schema.org. "Schema.org — Schemas for structured data on the internet." Accessed 2026. https://schema.org/

[2] Facebook. "The Open Graph Protocol." Accessed 2026. https://ogp.me/

[3] W3C. "JSON-LD 1.1 — A JSON-based Serialization for Linked Data." W3C Recommendation, 2020. https://www.w3.org/TR/json-ld11/

[4] Mozilla. "Readability.js — A standalone version of the readability library used for Firefox Reader View." Accessed 2026. https://github.com/mozilla/readability

[5] Honnibal, M., Montani, I., Van Landeghem, S., and Boyd, A. "spaCy: Industrial-strength Natural Language Processing in Python." Zenodo, 2020. https://spacy.io/

[6] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. "The Stanford CoreNLP Natural Language Processing Toolkit." In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014.

[7] OpenAI. "Function Calling and Structured Outputs." OpenAI API Documentation, 2024. https://platform.openai.com/docs/guides/function-calling

[8] Vrandecic, D. and Krotzsch, M. "Wikidata: A Free Collaborative Knowledge Base." *Communications of the ACM*, 57(10):78–85, 2014.

[9] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., Yih, W., Rocktaschel, T., Riedel, S., and Kiela, D. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[10] Anthropic. "Model Context Protocol." 2024. https://modelcontextprotocol.io/

[11] OpenAPI Initiative. "OpenAPI Specification v3.1.0." Accessed 2026. https://spec.openapis.org/oas/v3.1.0

[12] Sitemaps.org. "Sitemaps XML Format." Accessed 2026. https://www.sitemaps.org/protocol.html

[13] Kohlschutter, C., Fankhauser, P., and Nejdl, W. "Boilerplate Detection using Shallow Text Features." In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 441–450, 2010.

[14] Etzioni, O., Banko, M., Soderland, S., and Weld, D. S. "Open Information Extraction from the Web." *Communications of the ACM*, 51(12):68–74, 2008.

[15] Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. "FEVER: a Large-scale Dataset for Fact Extraction and VERification." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 809–819, 2018.

[16] W3C. "PROV-O: The PROV Ontology." W3C Recommendation, 2013. https://www.w3.org/TR/prov-o/

[17] Gao, Y., Xiong, Y., Dibia, V., Zhang, L., and Nyberg, E. "Retrieval-Augmented Generation for Large Language Models: A Survey." *arXiv preprint arXiv:2312.10997*, 2024.

[18] ISO 28500:2017. "Information and documentation — WARC file format." International Organization for Standardization, 2017.

[19] Common Crawl Foundation. "Common Crawl." Accessed 2026. https://commoncrawl.org/

[20] Kuhn, T., Willighagen, E., Groth, P., Seneviratne, O., Dumontier, M., and Mons, B. "Nanopublications: A Growing Resource of Provenance-Centric Scientific Linked Data." *IEEE Access*, 6:77966–77974, 2018.

[21] Freed, N. and Klensin, J. "RFC 6838: Media Type Specifications and Registration Procedures." IETF, 2013. https://www.rfc-editor.org/rfc/rfc6838

[22] Fielding, R., Nottingham, M., and Reschke, J. "RFC 9110: HTTP Semantics." IETF, 2022. https://www.rfc-editor.org/rfc/rfc9110

Appendix A: Example SDF Document


```
{
  "sdf_version": "0.2.0",
  "id": "sdf:43332a7148fb00833ad9b6d03738b036be06832532f8dd65b13888f5e35fcbce",
  "canonical_url": "https://blog.cloudflare.com/welcome-to-developer-week-2025/",
  "sdf": {
    "version": "0.2",
    "schema_uri": "https://sdfprotocol.org/schemas/sdf-document-0.2.schema.json",
    "dialect": "core",
    "features": ["temporal.v1", "links.v1"]
  },
  "parent_type": "article",
  "type": "news",
  "type_data": {},
  "language": "en",
  "source": {
    "url": "https://blog.cloudflare.com/welcome-to-developer-week-2025/",
    "title": "Welcome to Developer Week 2025",
    "author": "Rita Kozlov",
    "site_name": "The Cloudflare Blog",
    "published_at": "2025-04-06T18:00+01:00",
```

```
    "fetched_at": "2026-02-08T03:47:32.934Z"
  },
  "summary": {
    "brief": "Cloudflare is launching Developer Week 2025 to showcase new tools and produ
    "detailed": "The Cloudflare blog announces Developer Week 2025, highlighting advancer
    "key_points": [
      "Cloudflare is launching Developer Week 2025",
      "AI advancements are transforming software development",
      "Cloudflare provides a comprehensive platform for developers"
    ]
  },
  "entities": [
    { "name": "Developer Week 2025", "type": "event", "relevance": 1.0 },
    { "name": "Cloudflare", "type": "organization", "relevance": 1.0 },
    { "name": "Cloudflare Workers", "type": "technology", "relevance": 1.0 },
    { "name": "Rita Kozlov", "type": "person", "relevance": 0.8 }
  ],
  "claims": [
    {
      "statement": "Cloudflare is launching Developer Week 2025 featuring new tools for
      "confidence": 0.9,
      "type": "fact",
      "verification": { "status": "unverified", "priority": "medium" },
      "temporal": { "time_sensitivity": "high" }
    }
  ],
  "topics": ["developer week", "cloudflare", "ai development", "full-stack"],
  "metadata": {
    "word_count": 797,
    "reading_level": "intermediate",
    "content_quality": 0.8,
    "freshness": "current"
  },
  "temporal": {
    "published_at": "2025-04-06T18:00+01:00",
    "retrieved_at": "2026-02-08T03:47:32.934Z",
    "freshness": {
      "staleness_days": 308,
      "volatility": "fast",
      "recommended_recheck_after_days": 7,
      "reason_codes": ["frequently_changes"]
    }
  },
  "provenance": {
    "converter": "sdf-engine/0.2.0",
    "model_used": "qwen2.5:14b-instruct-q4_K_M",
    "conversion_confidence": 0.9,
    "content_hash": "sha256:43332a7148fb00833ad9b6d03738b036be06832532f8dd65b13888f5e35f
```

```
    "chain": [
      {
        "step_id": "s1",
        "actor": { "type": "service", "name": "fetcher", "version": "0.2.0" },
        "action": "fetch"
      },
      {
        "step_id": "s2",
        "actor": { "type": "service", "name": "analyzer", "version": "0.2.0" },
        "action": "extract"
      }
    ]
  }
}
```

Appendix B: Full Type Taxonomy (v0.2.0)

Parent	Leaf Types	MVP
article	news, blog, opinion, review, analysis, press_release	news, review
documentation	api_docs, support, tutorial, guide, changelog, faq	api_docs, support
commerce	product, job_posting, real_estate, service, auction	product
data	finance, sports, weather, scientific_dataset, report	finance
reference	encyclopedia, academic_paper, legal_document, recipe, medical_info, patent	–
discussion	forum_thread, q_and_a, social_post, comment_thread	q_and_a
code	repository, package, code_snippet, pull_request	repository
media	video, podcast, image_gallery, music_album	–
profile	person, organization, place	organization
event	conference, meetup, webinar, live_event, concert	conference

---

Appendix C: Figure Descriptions

Figure 1: SDF in the agent/web stack. Shows the baseline path (Agent -> HTML fetch -> boilerplate removal -> chunking -> extraction -> reasoning) vs. SDF path (Converter once per content

hash -> SDF cache/CDN -> many agents consume SDF directly). Annotates where costs shift and where provenance/temporal signals live.

Figure 2: SDF document model. Block diagram of the JSON envelope: header, identity, type fields, semantic core, type_data, temporal, provenance, extensions. Shows hierarchical type dispatch: parent_type -> leaf schema -> type_data validation. Shows aspect overlays as sidecar typed objects.

Figure 3: Two-pass conversion pipeline with normalization cascade. Updated pipeline: fetch -> normalize -> Pass1 classify (1.5B) -> type normalization cascade -> (optional 14B fallback) -> Pass2 extract (3B) -> validate -> emit. Shows the five normalization stages and branching logic for fallback.

Figure 4: Parent-type distribution histogram. Horizontal bar chart showing counts for each parent_type across 2,335 production documents (article=830, documentation=715, reference=412, discussion=250, ...). Highlights the long-tail distribution and the anomalous "recipe" parent type.

Figure 5: Latency-accuracy tradeoff by model configuration. Clustered bars or scatter plot comparing four configurations: (i) fine-tuned 1.5B+3B SmolLM3, (ii) fine-tuned 1.5B+3B Qwen, (iii) hybrid 1.5B+14B, (iv) 14B only. X-axis: total latency (ms, log scale), Y-axis: exact-match accuracy (%). Visually conveys the 4.1x speed gain with minimal accuracy loss.

Figure 6: Type normalization cascade flow. Sankey diagram with left nodes showing raw classifier outputs (74 unique types), middle nodes showing corrections at each cascade stage (enum enforcement, alias mapping, remap table, auto-fix, 14B fallback), and right node showing 100% valid types after normalization. Widths represent document counts.

SDF Protocol is an open standard. The specification, reference implementation, and schemas are available at sdfprotocol.org.