DRILLER at GenSIE 2026: Interpreting Extraction Semantics through Schema-Aware Prompting, Complementary Retrieval, and Heterogeneous Self-Consistency

Ariel González-Gómez[1,*,†], Daniel Alejandro Valdés-Pérez[1,†]

[1]University of Havana, Havana, Cuba

Abstract

DRILLER is the system submitted to GenSIE 2026 for Spanish schema-guided information extraction. GenSIE asks systems to extract grounded JSON objects from Spanish texts under JSON Schemas supplied at inference time. The main challenge is not only JSON validity, but field interpretation: names, descriptions, types, enum labels, nullability, and nesting can change from one instance to the next. DRILLER addresses this setting with three submitted pipelines built on a common schema-guided extraction architecture. `Schema-RAG` renders schemas in a model-readable form, constrains generation with a strict structured response format, and adds field-aware retrieved demonstrations. `Reasoned-RAG` adds temporary top-level reasoning/value wrappers so the model can assess local evidence before producing each final field value. `Mixed-SC` combines direct and reasoning-augmented trials, then selects a final prediction through schema-aware aggregation. Across the pipelines, DRILLER emphasizes schema interpretation, grounded extraction, conservative postprocessing, and compatibility with the model-agnostic hosted evaluation protocol.

Keywords

Spanish information extraction, schema-guided extraction, retrieval-augmented generation, structured generation, self-consistency

1. Introduction

Schema-guided information extraction requires systems to adapt to an output contract that may be unknown before inference time. The model receives a task-specific schema and must interpret each field for the current source text rather than extracting a fixed inventory of slots. This setting is increasingly common when language models are used behind APIs, form-filling workflows, or data integration tools: the schema is not just a validator, but part of the instruction. [1]

GenSIE 2026 formalizes this problem for Spanish information extraction [2]. Each instance provides a Spanish source text, an extraction instruction, and a target JSON Schema; the system must return a grounded JSON object that conforms to that schema. The task is part of IberLEF 2026 [3] and uses a hosted, model-agnostic evaluation protocol, so a submitted system must respect the model selected by the evaluator rather than relying on a privately chosen backend. DRILLER is therefore designed around bounded inference-time adaptation.

JSON validity is only one part of the task. A syntactically valid object can still be wrong if the system misreads field semantics: strings may require copied mentions, normalized values, or grounded summaries; enums may require mapping evidence to labels absent from the text; and nullable fields require deciding whether the passage supports a value. Arrays and nested objects add ambiguity about completeness and item identity.

With a fixed schema, field semantics could be learned from training data or encoded in task-specific rules. Because field names, descriptions, type constraints, enum labels, nullability, and nesting form a

IberLEF 2026, September 2026, León, Spain

*Corresponding author.

†These authors contributed equally.

ariel.gonzalez@matcom.uh.cu (A. González-Gómez); daniel.valdes@matcom.uh.cu (D. A. Valdés-Pérez)

0009-0005-6780-4307 (A. González-Gómez); 0000-0003-4472-3967 (D. A. Valdés-Pérez)

runtime semantic specification, DRILLER treats schema interpretation as part of extraction rather than as post-hoc validation.

DRILLER, originally named for *Dynamic Reasoning for Information Labeling and Literal Evidence Retrieval*, was submitted to the challenge as three pipelines. We refer to:

- `enriched-schema-rag` as `Schema-RAG`, the direct retrieval-augmented extractor.
- `enriched-inline-reasoning-rag` as `Reasoned-RAG`, which uses the same retrieval path but temporarily wraps top-level fields as reasoning/value objects during generation.
- `mixed-extractors-self-consistency-rag` as `Mixed-SC`, which combines direct and reasoning-augmented trials through heterogeneous self-consistency and schema-aware aggregation.

These aliases are used throughout the paper.

All three pipelines share a schema-guided extraction spine. DRILLER renders the target schema in a compact Pydantic-style view for the prompt, sends a strict JSON Schema response format to constrain generation, retrieves up to two field-aware demonstrations from a curated few-shot corpus, and applies deterministic postprocessing before returning the final object. The pipelines differ in whether they use a temporary reasoning scaffold and whether they aggregate multiple candidates.

The readable schema presented in the prompt foregrounds field names, descriptions, types, enums, nullability, and nesting. The generation schema constrains the model to return a complete JSON object. This separation lets DRILLER present the schema in a typed, class-like form while preserving the evaluator's original JSON shape.

Field-aware few-shot retrieval selects examples by schema relevance, not by factual overlap with the current source. DRILLER first looks for demonstrations with the same normalized schema. When exact structural matches are unavailable, it falls back to field-level retrieval with type-compatible matching and a complementary pair objective. Retrieved examples illustrate extraction behavior, including contrasts such as present versus absent values, but their content is not evidence for the current instance.

`Reasoned-RAG` represents each top-level field during generation as an object with a `reasoning` string and a `value` of the original field type, targeting fields whose values require evidence assessment, null decisions, or schema-specific label mapping. After generation, the reasoning strings are discarded and only the values are returned. `Mixed-SC` then uses both direct and reasoned extractors as heterogeneous candidate generators, aggregating their final-shape JSON outputs with recursive schema-aware rules for scalars, nulls, objects, and arrays.

The contributions of this work are:

- a modular schema-guided extraction architecture for variable JSON Schemas in Spanish information extraction;
- schema-readable prompting paired with strict structured generation, separating schema interpretation from output-interface enforcement;
- field-aware few-shot retrieval with same-schema matching, type-compatible field fallback, and complementary example selection;
- a temporary top-level reasoning/value transformation that supports field-local evidence assessment while preserving the final schema interface;
- heterogeneous self-consistency over direct and reasoning-augmented RAG extractors with conservative schema-aware postprocessing and aggregation.

2. Background and Related Work

Schema-guided information extraction treats the schema as an instance-level specification of extraction targets rather than a fixed ontology. In GenSIE, the required JSON Schema specifies not only the output form, but also the extraction subtasks induced by field names, descriptions, types, enum labels, nullability, arrays, and nesting [2]. Related schema-aware and universal information extraction work

has studied how extraction behavior can be conditioned on schema information, including approaches based on schema-aware extraction modules and code-style schema representations [1, 4].

Few-shot prompting addresses how task behavior can be provided at inference time without updating model parameters [5]. However, in-context performance depends strongly on which examples are selected, motivating retrieval-based prompt selection and semantic-similarity example retrieval [6, 7]. Sentence embeddings provide a standard mechanism for comparing textual representations by semantic similarity [8]. This differs from retrieval-augmented generation over external factual evidence [9]: DRILLER retrieves demonstrations for dynamic few-shot prompting, not evidence for the current instance. In information extraction, schema-aware reference-as-prompt methods provide a closer precedent for using retrieved schema-conditioned examples as prompts [10].

Structured generation methods constrain language-model outputs to formal interfaces such as grammars, schemas, or guided output spaces [11, 12]. These methods address a necessary part of schema-guided extraction: the output must be parseable and compatible with the requested structure. At the same time, structural validity and extraction correctness remain distinct. A schema-valid JSON object can still contain an unsupported value, a wrong enum label, an incorrect null decision, or an incomplete array.

Reasoning-oriented prompting, including chain-of-thought prompting, elicits intermediate reasoning before final answers [13]. In structured information extraction, related work has also separated intermediate natural-language content generation from the subsequent organization of that content into the desired structured format [14]. Open-ended chain-of-thought is not the interface DRILLER returns: its scaffold is inline and field-local. Each top-level field is temporarily generated as a structured rationale/value pair, so the model must assess the local evidence immediately before committing to that field's value. Because the scaffold is part of the temporary response schema, this requirement can be enforced independently for each top-level field, while the value slot remains constrained by the field's original type.

Self-consistency samples multiple reasoning paths and selects the most consistent answer rather than relying on one generation [15]. For structured extraction, the analogous problem is not only answer selection but also how to combine candidate JSON objects whose fields, arrays, or nested objects may disagree. Minimum Bayes risk decoding provides a general decision-theoretic view of selecting outputs by expected loss or utility over a candidate set [16]. DRILLER uses this idea only in an MBR-style array-selection heuristic based on agreement among candidate arrays.

3. Task Setting

GenSIE is a Spanish schema-guided information extraction task [2]. Each instance provides a source text, a natural-language extraction instruction, and a target JSON Schema. The source text is the only evidence for the answer. The instruction states the extraction goal, and the schema defines the output structure through field names, descriptions, primitive types, objects, arrays, enum labels, nullability, and required properties.

The required output is a JSON object grounded in the source text and conforming to the target schema. Conformance includes producing the expected object shape, using valid primitive and enum values, and representing nested structures and arrays according to the schema. Grounding means that values should be supported by the current passage rather than external knowledge, retrieved examples, or plausible background assumptions.

Schemas vary across instances, so the extractor cannot rely on a fixed ontology or stable slot inventory. The same textual mention may be relevant for one schema and irrelevant for another; the same JSON type may correspond to different behaviors depending on the field description. The schema must therefore be interpreted dynamically at inference time.

The official protocol also constrains inference cost. The submission guidelines describe token and wall-clock budgets as soft test-set averages: a target of 32K total input-plus-output tokens per instance, including reasoning, retries, and self-corrections, and a target of 60 seconds of user-code time per

instance [17]. These constraints mean that a system cannot assume arbitrarily many model calls, unbounded retries, or unlimited reasoning traces; inference-time methods must be organized as a bounded procedure.

The task description emphasizes schema generalization: private evaluation may include both schemas that appeared during development and entirely new schemas, without reusing development source documents [18]. Exact or near-exact schema reuse is therefore a realistic evaluation condition, but not one that can be assumed for every instance. DRILLER's same-schema retrieval is a deliberate inference-time strategy: it first searches for structurally identical schema demonstrations before falling back to field-level analogies when no suitable same-schema references are available.

Missing information must be handled conservatively. When the requested value is not supported by the source text and the schema permits null, the system should return JSON `null`. If evidence is present, returning `null` is also an error. DRILLER therefore treats nullability as a schema-guided evidence decision rather than a generic fallback.

Official dataset construction, scoring, ranking, and evaluation protocol details are specified in the GenSIE overview paper. We focus on the submitted DRILLER pipelines and their inference-time mechanisms for producing grounded schema-conformant JSON under that protocol.

4. System Overview

The submitted DRILLER system comprises three pipelines: `Schema-RAG`, `Reasoned-RAG`, and `Mixed-SC`. They share a schema-guided extraction architecture and differ in reasoning scaffolds, trial count, and final decision rule. Their submitted interface names are `enriched-schema-rag`, `enriched-inline-reasoning-rag`, and `mixed-extractors-self-consistency-rag`. Non-registered experimental variants are outside the submitted system.

Table 1 summarizes the final pipelines, all of which use retrieval-augmented prompting over local demonstrations [9] and strict structured JSON generation [11, 19].

Table 1
Submitted DRILLER pipelines.

Alias	Schema view	Reasoning wrapper	Requested trials	Final output
`Schema-RAG`	Pydantic	No	1	Direct
`Reasoned-RAG`	Reasoned Pydantic	Top-level	1	Unwrap + direct
`Mixed-SC`	Direct and reasoned	Reasoned trials	up to 4	Schema-aware SC

Figure 1 traces the shared path from a GenSIE instance to a final JSON object. Given the Spanish source text, extraction instruction, and target JSON Schema, DRILLER renders the schema for prompting, retrieves local demonstrations, constructs a Spanish extraction prompt, requests a structured JSON object from the hosted model, parses and normalizes the output, and optionally aggregates multiple candidates. The returned object always has the original challenge schema shape.

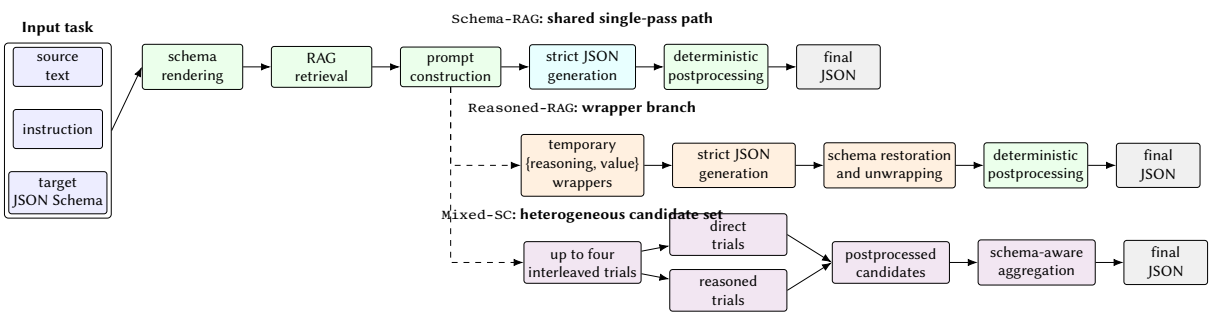

Figure 1: Architecture of the submitted DRILLER pipelines.

The prompt presents the instance instruction, grounding rules, retrieved examples when available, a Pydantic version of the target schema, and the source text. The rules state that the source text is the only evidence, examples are demonstrations rather than facts for the new instance, enum labels must be used exactly, and unsupported nullable fields should be returned as JSON `null`. This organization keeps the schema and passage close to generation while making the evidence boundary explicit.

In direct mode, schema rendering shows the JSON Schema as a compact Pydantic-style model: fields become typed attributes, descriptions remain near the fields they define, arrays and nested objects are readable as typed structures, and enums are displayed as literal alternatives. This prompt view is not the formal output contract. The model request also includes a strict JSON Schema response format derived from the target schema, and the final answer is checked against the evaluator-facing schema shape after postprocessing.

Each extraction can include up to two local few-shot demonstrations selected for schema relevance. Same-schema retrieval is used when examples have the same normalized structural schema as the current task; otherwise, DRILLER falls back to field-level matching over field names, descriptions, type classes, and complementary coverage. The selected examples are rendered in the same output style as the current extractor: direct examples for `Schema-RAG`, reasoning/value examples for `Reasoned-RAG`, and trial-specific rendering inside `Mixed-SC`.

Deterministic postprocessing parses JSON, cleans common Unicode artifacts, normalizes strings to *Normalization Form C* (NFC), removes temporary reasoning wrappers when present, and converts the literal string `"null"` to JSON `null` only at schema positions where null is allowed. It does not infer missing facts, repair unsupported values, or reinterpret the source text after generation.

`Schema-RAG` uses the Pydantic-style prompt schema, retrieves demonstrations, requests a strict JSON object, and returns the parsed and normalized object. Because no reasoning wrapper is used, the generated top-level fields already match the final output interface.

`Reasoned-RAG` follows the same path but temporarily transforms each top-level field into an object with `reasoning` and `value` slots during generation. The prompt schema exposes this as a reasoned value, and the response schema requires the wrapper structure. After generation, DRILLER discards the reasoning strings and returns only the values in the original schema shape.

`Mixed-SC` moves from a single extraction to a heterogeneous candidate set. It requests up to four budget-aware trials: two direct `Schema-RAG`-style attempts and two reasoning-augmented `Reasoned-RAG`-style attempts in an interleaved plan. Each completed trial performs its own retrieval, prompt construction, strict generation, and postprocessing. Aggregation receives final-shape JSON candidates, not raw model messages or reasoning traces, and combines them recursively using schema-aware rules for scalars, nulls, objects, and arrays.

5. Schema-Guided Prompting

DRILLER treats the schema as part of the task semantics, not only as a formatting constraint: the prompt helps the model interpret each field, while the model request constrains the response to a JSON object.

The extraction prompt asks the model to use the current source text as the only evidence, follow the extraction instruction, and interpret field meanings through the target schema. Retrieved examples are demonstrations of extraction behavior, not evidence for the current input. The answer must be only the requested JSON object, without markdown fences, explanations, or extra text.

The standard prompt sets the role of a precise Spanish information extractor, gives rules for grounding, full schema coverage, exact enum labels, conservative null handling, and requested normalization, then inserts retrieved examples. It renders the current schema in a model-readable form immediately before the source text, keeping the schema and evidence close to generation.

Raw JSON Schema is well suited for validation, but it can be visually heavy inside a prompt. Nested property dictionaries, separate required lists, repeated keywords, and nullable type arrays can obscure the field descriptions that carry extraction semantics. DRILLER therefore renders the schema in a Pydantic-style view. The view is not an executable program and does not replace the target JSON

Schema; it is a compact and more readable representation of the current extraction contract.

Object schemas become class-like structures and properties appear as typed attributes. Primitive fields are rendered as familiar scalar types, arrays as list types, enums as literal alternatives, and nested objects as nested or referenced model types. Nullability is shown near the field rather than separated from the definition. Field descriptions are preserved because they often determine the correct behavior: the same type annotation can require copying, normalization, classification, or a short grounded summary depending on the description.

Figure 2 shows the transformation on a compact excerpt from the `cultural_media_001` development schema. The raw JSON Schema excerpt carries the validation-oriented structure: a definition for the enum, property dictionaries, and nullable alternatives. The right panel shows two Pydantic-style renderings for the same selected fields: the plain compact renderer and the reasoned view. Both define the prompt-only alias `Nullable[T] = T | null`; the reasoned view additionally defines the generic `Reasoned[T]` wrapper and changes each root field from T to `Reasoned[T]`. Nested structures remain inside the `value` slot rather than being wrapped recursively.

Raw JSON Schema view

```
{
  "$defs": {
    "MediumType": {
      "enum": [
        "MOVIE",
        "BOOK",
        "TV_SERIES",
        "VIDEO_GAME"
      ],
      "title": "MediumType",
      "type": "string"
    }
  },
  "properties": {
    "medium": {
      "$ref": "#/$defs/MediumType",
      "description": "The type of media being
     reviewed"
    },
    "director": {
      "anyOf": [
        {
          "type": "string"
        },
        {
          "type": "null"
        }
      ],
      "default": null,
      "description": "The person who directed
     the film or series",
      "title": "Director"
    }
  }
}
```

Plain compact Pydantic renderer

```
Nullable[T] = T | null

MediumType = Literal["MOVIE", "BOOK", "
     TV_SERIES", "VIDEO_GAME"]

class Output(BaseModel):
    medium: MediumType = Field(..., description
     ="The type of media being reviewed")
    director: Nullable[str] = Field(None,
     description="The person who directed the
     film or series")
```

Reasoned Pydantic view

```
Nullable[T] = T | null

class Reasoned[T](BaseModel):
    reasoning: str
    value: T

MediumType = Literal["MOVIE", "BOOK", "
     TV_SERIES", "VIDEO_GAME"]

class Output(BaseModel):
    medium: Reasoned[MediumType] = Field(
     description="The type of media being
     reviewed")
    director: Reasoned[Nullable[str]] = Field(
     description="The person who directed the
     film or series")
```

Figure 2: Schema transition example from a reduced `cultural_media_001`. The left panel shows a raw JSON Schema excerpt from the task's `target_schema`; the right panel shows the corresponding selected-field excerpts emitted by the plain compact and reasoned Pydantic-style prompt renderers. The excerpt keeps a string-valued enum field, a nullable field, and field descriptions while omitting root metadata and unrelated fields for space.

For the 13 unique target-schema structures observed in the development set, deduplicated by canonicalized `target_schema` JSON, we measured schema-text footprint. The raw baseline is the indented JSON Schema text produced by the prompt schema renderer. The direct and top-level-reasoned Pydantic rows use the actual schema-view code paths selected by `SchemaPromptMode.PYDANTIC` and `SchemaPromptMode.REASONED_PYDANTIC`. We also report `render_plain_pydantic_schema`,

the same compact prompt-only renderer family used by the reasoned view, but without `Reasoned[T]` wrappers; this row isolates wrapper overhead from unrelated prelude differences. Size is estimated as $\lceil \text{characters}/4 \rceil$, matching the character-based token fallback used by DRILLER's trial-budget planner. The measurement estimates prompt footprint only, not model-specific tokenization.

Table 2
Schema rendering footprint over 13 unique development schemas. Values are estimated prompt tokens computed as $\lceil \text{characters}/4 \rceil$.

Rendering	Mean	Median	Min	Max	Mean vs. raw
Raw JSON Schema	570	660	109	886	–
Direct Pydantic view	276	310	67	385	-51.6%
Plain compact Pydantic renderer	249	283	41	357	-56.3%
Reasoned Pydantic view	272	306	58	378	-52.2%

Table 2 describes schema text alone, not the complete prompt or completion cost. On these schemas, the class-like view roughly halves the prompt-side schema footprint relative to indented raw JSON Schema. The actual reasoned view is slightly shorter than the actual direct Pydantic view because the direct code path includes import-style boilerplate, while the reasoned prompt renderer uses a prompt-only prelude. In the like-for-like renderer comparison, top-level reasoning wrappers increase the mean footprint from 249 to 272 estimated tokens. The reasoned view remains in the same range because the submitted reasoning mode wraps only top-level fields.

Table 3 summarizes the three schema roles in DRILLER. The same target schema is transformed differently when shown to the model, used to constrain decoding, or returned to the evaluator.

Table 3
Schema roles in DRILLER. The prompt schema supports model comprehension; the generation schema constrains decoding; the final challenge schema defines the returned interface.

Schema role	Where it is used	Function in the system
Prompt schema	Text shown inside the extraction prompt	Pydantic-style or reasoned Pydantic-style rendering that helps the model interpret field names, descriptions, types, enums, nullability, and nesting.
Generation schema	Structured response format sent with the model request	Strict JSON Schema used to constrain the generated object; in reasoning mode, temporary wrappers are inserted before generation.
Final challenge schema	Shape expected by the evaluator	Original task schema after schema restoration when needed and deterministic cleanup; reasoning scaffolds and prompt-only conventions are not returned.

The prompt schema can use class-like declarations, readable nullable notation, and other prompt-only conventions because it communicates the schema to the model. The generation schema has a different role: DRILLER attaches a strict JSON Schema response format to the model request so decoding is constrained toward a JSON object compatible with the requested structure.

For the submitted non-baseline pipelines, declared object properties are required during generation. This reduces field omissions, a common failure mode when a model understands the task but leaves out keys. Requiring keys does not require every value to be substantive. If the target schema permits null, the generated object may include the key with JSON `null`. The final challenge schema remains the original schema shape; generation-time requiredness is an inference constraint, not permission to fabricate unsupported content.

In direct mode, used by `Schema-RAG`, the prompt schema is ordinary Pydantic-style rendering and the model produces final values directly. In reasoning mode, used by `Reasoned-RAG` and the reasoning-

augmented trials of `Mixed-SC`, each top-level field is shown as a reasoned value. The temporary generation schema requires an object with `reasoning` and `value` fields for each top-level slot. The `value` field retains the original type, including enums, arrays, objects, and nullable alternatives.

The reasoning transformation changes the temporary interface without asking for free-form explanation. The model has a structured place to state local evidence or absence before committing to a field value, and schema restoration removes the reasoning text before returning the final object. Direct and reasoned prompting therefore share the same final objective while exposing different intermediate interfaces.

In the general RAG layout, each example carries enough schema context for the model to understand why its output follows from its own source text and schema. In the same-schema layout, selected examples share the normalized target schema, so DRILLER can render the shared schema once and show compact example inputs and outputs. This saves prompt size and lets the examples emphasize contrasting field outcomes under the same structure.

Examples can show how fields are populated, when null or empty arrays are appropriate, and how enum labels or nested structures behave, but their values are not evidence for the current instance. In reasoning mode, retrieved examples are rendered with the same temporary `{reasoning, value}` structure expected from the current extractor; in direct mode, examples use final field values. `Mixed-SC` applies the appropriate layout separately inside each trial.

Schema-guided prompting in DRILLER combines readable schema presentation, explicit grounded-extraction rules, strict structured generation, and retrieval-aware layout. It makes schema interpretation explicit while returning a complete JSON object compatible with the evaluator-facing schema.

6. Field-Aware Few-Shot Retrieval

DRILLER retrieves examples for schema interpretation rather than external factual evidence: in a variable-schema task, semantic similarity between source texts is not enough. Two passages may be topically close but ask for different fields, while two different domains may instantiate similar extraction behavior. The submitted pipelines retrieve demonstrations using schema structure, extraction instruction, field descriptions, type compatibility, and complementary output behavior. `Schema-RAG`, `Reasoned-RAG`, and every trial of `Mixed-SC` use the same retrieval path.

Figure 3 summarizes the two retrieval paths used by DRILLER.

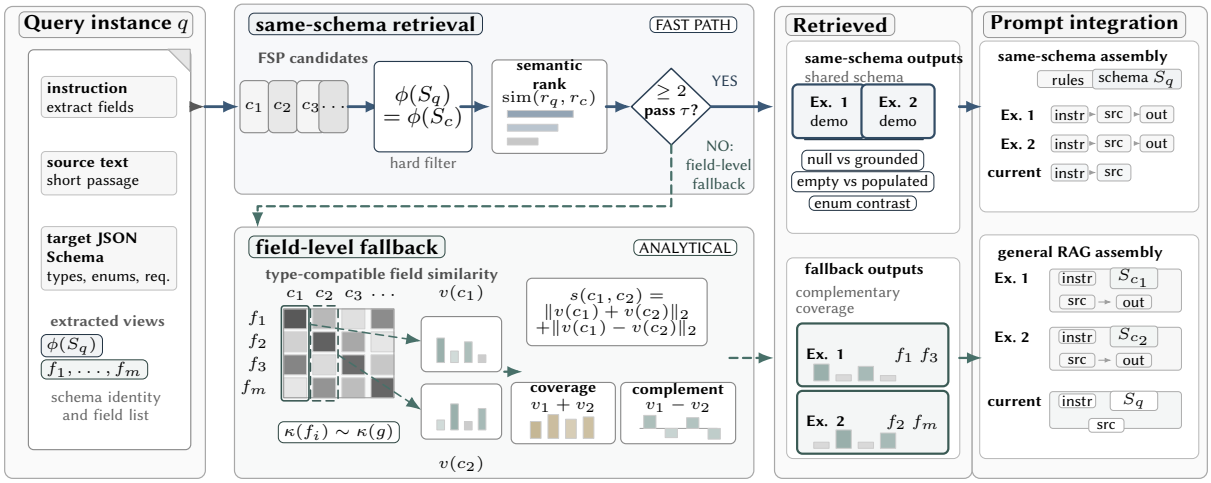

Figure 3: Two-stage retrieval in DRILLER. The system first attempts same-schema retrieval by filtering candidate demonstrations with a normalized schema fingerprint and ranking surviving cases semantically. A decision node routes directly to same-schema demonstrations when at least two cases pass the threshold; otherwise, field-level fallback scores candidates by type-compatible field similarity and selects a complementary pair. The selected demonstrations are then rendered with either the compact same-schema prompt layout or the general RAG layout.

6.1. Curated Demonstration Corpus

The retrieval pool is a local few-shot prompting (FSP) corpus of structured extraction demonstrations [5]. Each case contains a Spanish source text, an extraction instruction, a target schema, expected values, tags describing covered extraction phenomena, and field-level rationales or descriptions that support rendering examples in either direct or reasoning-augmented form. The corpus is synthetic and curated rather than mined from an external collection.

We used the GenSIE development set in the corpus design to study schema structure, source-text style, vocabulary, noise patterns, and recurring ambiguities, rather than reusing development instances as demonstrations. We also manually curated a small development-time evaluation subset, reviewing up to two instances per task family. Those curated instances served as design references only. Direct reuse would not provide systematic complementary coverage of field dualities and would contaminate evaluation on the same curated subset, since retrieval could return the instance being solved.

We constructed the final FSP corpus as a compact library of synthetic and manually curated structured cases, organized by development task family and schema family. The corpus used an author-in-the-loop workflow assisted by an agentic LLM coding assistant. The assistant helped inspect candidate task families, draft complementary-case proposals, generate provisional source texts and structured resources, and run structural checks. The authors reviewed and corrected the proposed dualities, source texts, expected outputs, reasoning fields, enriched descriptions, and final JSON resources before inclusion.

The final extraction corpus contains two cases for each of the 22 schema or task families represented in the curated development setting, for 44 extraction cases in the local retrieval pool. The families cover a range of domains and structures, but the corpus is not intended to be a large topical nearest-neighbor database. It is a compact library of contrasted demonstrations: the useful unit is often a pair of cases showing how the same or related fields behave under different evidence conditions.

A nullable field benefits from both grounded non-null values and grounded absence. An array field benefits from examples where the correct list is populated and examples where it is empty. Enum fields benefit from contrasting labels, especially when labels are inferred rather than copied literally. Numeric and date fields may require exact copying in one case and normalization in another. String fields may require direct mention extraction, near-verbatim evidence, or a short grounded summary depending on the field description.

After complementary cases had been proposed and reviewed, we added enriched field descriptions as compact Spanish prompt annotations. They explain how to extract a field and warn about traps observed during development-example review or synthetic case construction, such as distractor dates, nearby but semantically wrong entities, aliases, or conditions under which a nullable field should remain `null`. These descriptions clarify schema semantics for same-schema prompting; they do not add new instance evidence. They appear only in the prompt-side schema view. The target JSON Schema, strict generation schema, and evaluator-facing output schema remain unchanged.

In reasoning-mode examples, each field can render `field_asks`, `relevant_fragments`, and `final_value` as a local evidence protocol. Cited fragments support the extraction decision rather than merely repeating the answer string. They stay short while preserving the context needed to justify the value: accepted candidates show why the value belongs to the requested field, and rejected or `null` cases cite the closest reviewed context and state which required assertion is missing. Enum reasoning considers the allowed labels and justifies the selected label from relevant fragments. Array reasoning may cite a complete list once, or cite compactly around individual items, so examples support recall without bloating the prompt. FSP examples therefore demonstrate grounding behavior as well as output formatting.

Table 4 gives one compact example from the `technical_software` family. The two cases share the same schema but expose complementary regimes: one case grounds populated arrays and leaves several adversarial nullable fields unsupported, while the other inverts the pattern with empty arrays, alias resolution, exact date normalization, an explicit stable-installer checksum, and explicit corporate metadata.

Table 4
Complementary FSP pair for the `technical_software` schema family.

Field or contrast	*Lince Editor* case	*QuantaPack Pro* case
`official_name` and `app_category`	Direct product name; enum maps to `IDE_EDITOR`.	Abbreviated heading must resolve to full name; enum maps to `ARCHIVER`.
`platforms` and `features`	Populated arrays with named operating systems and technical capabilities.	Empty arrays: installers and downloads are mentioned, but no supported OS or concrete feature list.
`exact_release_date`	`null`: only year-level or relative release evidence is present.	Exact first-release date normalized to `DD/MM/YYYY`.
`latest_stable_version_sha256`	`null`: PGP signatures and a nightly commit hash are distractors.	Explicit SHA256 for the stable installer.
`current_ceo_name`	`null`: founder, idea author, and coordinating team are not CEO evidence.	Explicit current CEO of the developing company.
Pair lesson	Extract grounded scalar values and populated arrays, but abstain for nullable fields with only distractor evidence.	Resolve aliases, keep unsupported arrays empty, and fill nullable fields only when the source states the required evidence.

Coverage includes null versus non-null values, present versus absent evidence, enum alternatives, empty versus populated arrays, arrays of simple values, arrays of objects, nested objects, direct strings, summary strings, boolean inference, numeric normalization, date normalization, bounded scores, long evidence spans, and distractor mentions. These contrasts matter because few-shot examples can bias the model when they show only one outcome regime. Complementary demonstrations teach conditional behavior: extract when evidence exists, abstain when it does not, normalize only when requested, and respect the requested type and label set.

6.2. Same-Schema Retrieval

At runtime, DRILLER first tries to retrieve examples with the same normalized schema as the current task. Let S_q be the target schema of query instance q, and let S_c be the schema of a candidate case c. DRILLER computes a normalized schema fingerprint $\phi(S)$ by canonicalizing the JSON representation while preserving the structural extraction contract: field names, object structure, array structure, primitive types, enum alternatives, required properties, and nesting. Descriptions are neutralized so that prompt wording refinements do not change schema identity. Order-insensitive elements such as `required`, `enum`, and array-valued `type` are sorted before serialization. Same-schema candidates satisfy the hard constraint

$$\phi(S_q) = \phi(S_c).$$

The fingerprint treats structural identity as a filter rather than as a soft ranking feature. A medical passage and a technical passage might both contain names, dates, arrays, and classifications without sharing an extraction form. Conversely, two source texts from the same schema family may require the same field interpretation even when their values differ.

After filtering, DRILLER builds a global textual representation $r(q)$ of the current task from the task identifier, extraction instruction, and textual representations of top-level fields. Candidate cases are represented analogously as $r(c)$. These representations are embedded and compared by cosine similarity,

$$\mathrm{sim}(r(q), r(c)).$$

The submitted system uses a same-schema threshold $\tau = 0.6$. If at least two fingerprint-matching candidates satisfy $\mathrm{sim}(r(q), r(c)) \geq \tau$, DRILLER selects the top candidates by similarity and marks them as same-schema demonstrations. If fewer than two candidates pass this test, retrieval continues with the field-level fallback.

Same-schema examples show the exact output contract under different source texts. They clarify what counts as evidence for a field, when null or an empty list is correct, and how enum labels or nested

structures are populated. They also enable a compact prompt layout: the shared schema is rendered once, and examples focus on instruction, source text, and expected output rather than repeating identical schema blocks.

Same-schema mode can also enrich the prompt-side schema view with curated field descriptions drawn from the selected cases. These enriched descriptions add task and domain guidance such as enum interpretation, null traps, or distractor patterns to the schema text shown to the model. They do not modify the target JSON Schema, the strict generation schema, or the final evaluator-facing schema.

6.3. Field-Level Retrieval Fallback

When same-schema retrieval does not produce enough candidates, DRILLER falls back to field-level retrieval. Let the query top-level fields be $F_q = \{f_1, \ldots, f_m\}$, in schema order, and let F_c be the top-level fields of candidate case c. Each field f is mapped to a textual representation $t(f)$ that combines its field name, readable type, and description, for example "`name (type): description`". The retriever also assigns a coarse type class $\kappa(f)$, ignoring nullability.

The coarse classes used by the retriever are `string`, `boolean`, `numeric` for both integer and number fields, `enum`, `object`, and recursively typed arrays such as `array[string]`, `array[numeric]`, `array[enum]`, and `array[object]`. A fallback `any` class is available for unsupported or underspecified schemas. We write $\kappa(f_i) \sim \kappa(g)$ for compatibility; in the submitted implementation this relation is equality of the nullability-insensitive coarse classes. Thus nullable and non-nullable variants of the same base type can match, while an array of objects is not aligned with a scalar summary merely because their descriptions share topical words.

For each candidate c, DRILLER embeds the field texts $e(t(f))$ and computes, for every query field f_i, the best compatible field match in the candidate:

$$a_i(c) = \max\left(0, \max_{g \in F_c,\ \kappa(g) \sim \kappa(f_i)} \cos(e(t(f_i)), e(t(g)))\right),$$

with $a_i(c) = 0$ when no compatible candidate field exists. The candidate is therefore represented by a field-similarity vector aligned with the current query schema,

$$v(c) = (a_1(c), \ldots, a_m(c)).$$

A candidate's individual relevance can be summarized by $\|v(c)\|_2$, but two-example selection uses a pair-level objective over these same vectors.

The type-aware vector prevents topical retrieval from ignoring output shape. It avoids matching semantically similar descriptions with incompatible JSON structures, allows nullable and non-nullable variants of the same base type to reinforce each other, and shows whether a candidate is broadly useful or strong for only part of the schema. Since retrieval selects a pair of demonstrations, two individually imperfect candidates can be preferable when they cover different field behaviors.

If embedding retrieval fails or produces no usable candidates, DRILLER uses a lexical/schema fallback. It scores candidates with explicit features such as exact normalized schema matches, compatible field fingerprints, tag overlap, field-name overlap, shared terms, and domain-term overlap, with a small penalty for long example texts. This fallback preserves schema-aware retrieval when embeddings are unavailable or uninformative.

6.4. Complementary Pair Selection

The submitted RAG path retrieves up to two examples. In the field-level fallback, DRILLER selects a pair with a complementarity-aware objective rather than choosing the two highest individual scores. For two candidate demonstrations c_1 and c_2, with field-similarity vectors $v(c_1)$ and $v(c_2)$ defined above, the pair score is

$$s(c_1, c_2) = \|v(c_1) + v(c_2)\|_2 + \|v(c_1) - v(c_2)\|_2.$$

The first term, $\|v(c_1) + v(c_2)\|_2$, rewards joint relevance and field coverage. If both examples provide useful analogues for the current schema, their combined vector has large magnitude. This term aligns with the ordinary retrieval goal that demonstrations should be relevant to the task.

The second term, $\|v(c_1) - v(c_2)\|_2$, rewards complementarity. If two candidates are strong on the same fields and weak on the same fields, their difference vector is small. If one is strong where the other is weak, the difference grows. This discourages near-duplicate demonstrations that reinforce one behavior while leaving other fields unexplained.

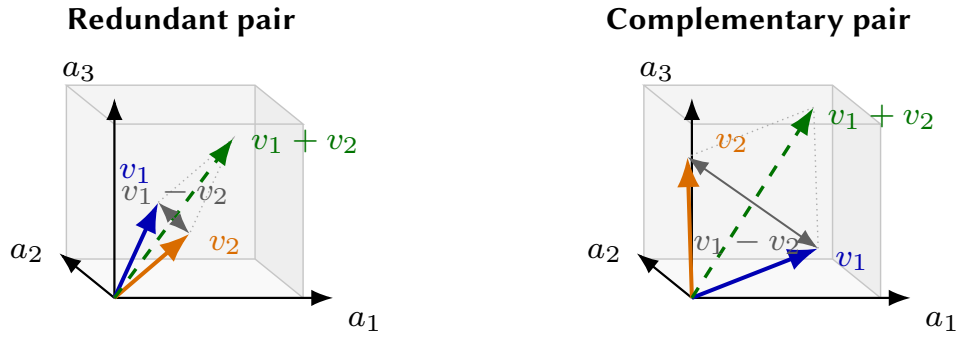

Figure 4: Field-aware pair selection as a vector objective. Each axis is a query-field similarity component a_i; actual retrieval vectors are m-dimensional. A redundant pair concentrates evidence in similar directions, so $v_1 - v_2$ is short. A complementary pair covers different field dimensions, increasing both the joint vector $v_1 + v_2$ and the difference vector $v_1 - v_2$ used by the pair score.

Two examples from different domains can still be redundant if both show easy scalar fields and neither illustrates nulls, arrays, or enum decisions. Conversely, two examples from a related family may be complementary if one illustrates absence and empty arrays while the other illustrates populated nested objects and label mapping. Tie-breaking favors stronger individual relevance and stable resource order, but the core decision is pair-level.

6.5. Prompt Integration

When examples are related but not all same-schema matches, the general RAG layout renders each example as a self-contained demonstration with its instruction, schema context, source text, and expected output. This gives the model enough context to see how the example output follows from that example's schema before using it analogically.

The same-schema layout is used when all selected examples share the normalized target schema. The shared schema is rendered once, alongside the role instruction, extraction rules, root description when available, and guidance that examples are demonstrations rather than evidence. The examples then omit repeated schema blocks and focus on source-to-output behavior under the common structure. This reduces prompt length and makes contrasting field outcomes more salient.

In `Schema-RAG`, examples are rendered in direct final-value form, matching the object shape returned after generation. In `Reasoned-RAG`, examples are rendered with top-level `{reasoning, value}` objects because the temporary generation interface requires that structure. The reasoning strings demonstrate a local evidence protocol, while the `value` slots show the values that will later be unwrapped.

`Mixed-SC` uses the same retrieval machinery inside each trial. Direct trials use the `Schema-RAG` rendering, and reasoning-augmented trials use the `Reasoned-RAG` rendering. Each trial performs retrieval independently. Aggregation receives only postprocessed JSON candidates in the final schema shape.

7. Reasoning-Augmented Extraction

`Reasoned-RAG` adds a temporary reasoning interface to the shared extraction pipeline. Some GenSIE fields require local evidence assessment before a value can be produced: the model may need to identify relevant fragments, decide whether evidence is absent, normalize a phrase, or map cues to an enum label. The reasoned interface gives each top-level field a structured place for that intermediate decision while preserving the final JSON interface.

In `Reasoned-RAG`, each top-level field of the target object is temporarily wrapped as an object with two properties:

```
{
  "reasoning": "...",
  "value": ...
}
```

The `reasoning` property is a string; the `value` property keeps the original schema of the wrapped field. If the original field is an enum, the `value` must be one of the enum labels; if it is an array or object, the nested structure remains there; if it is nullable, the `value` may be JSON `null` when the schema permits it. The wrapper changes the temporary generation shape, not the semantic target.

The submitted reasoning mode wraps root object properties only, not every nested property recursively. If a top-level field is itself an object or an array of objects, the nested content remains inside `value` according to the original field schema. This keeps the prompt and response format manageable while asking the model to deliberate at the level of the main requested slots.

In the prompt schema, each top-level field of type T is shown as a `Reasoned[T]` value. The extraction instructions do not ask for arbitrary chain-of-thought. They require each `reasoning` string to use three literal Spanish section labels, in this order:

```
EL CAMPO PIDE: ...
FRAGMENTOS RELEVANTES: ...
VALOR FINAL: ...
```

The `EL CAMPO PIDE` section makes the model restate the local schema request before filling the slot. This forces interpretation of the field name, type, enum alternatives, nullability, and description, which is especially useful when two fields share the same JSON type but ask for different extraction behavior. The `FRAGMENTOS RELEVANTES` section bounds the evidence used for the decision. It asks for only the source fragments that decide the field, or an explicit statement of absence when the field is unsupported. This discourages importing evidence from few-shot examples or from general knowledge. The `VALOR FINAL` section connects the evidence to the schema-compatible value that will be placed in `value`: an exact enum label, a normalized scalar, an array, an object, an empty array, or JSON `null` when permitted.

The reasoning interface is part of the strict temporary response schema, so each wrapped top-level field must contain both `reasoning` and `value`. This keeps reasoning inside the JSON object and keeps the final value constrained by the original field type. The wrapper adds a field-local explanation channel without relaxing the schema constraints on the answer.

After generation, DRILLER restores the original schema. Each top-level object of the form `{reasoning, value}` is replaced by its `value`, returning the object to the evaluator-facing schema. This wrapper removal is the inverse of the temporary schema transformation, not semantic post-hoc repair. The reasoning strings may be retained as trace metadata, but they are discarded before the prediction is returned or passed to self-consistency aggregation. If the expected wrapper structure is malformed and cannot be reliably restored, the extraction record is treated as invalid.

In reasoning-augmented RAG rendering, examples retrieved for `Reasoned-RAG` use the same temporary output format expected from the current extractor. A direct example with final field values would be inconsistent with a prompt that asks for top-level `{reasoning, value}` objects. Reasoning-mode

examples therefore demonstrate both the evidence protocol and the placement of the final answer in the `value` slot. The same labels, EL CAMPO PIDE, FRAGMENTOS RELEVANTES, and VALOR FINAL, appear inside example reasoning strings, so the model sees field interpretation, evidence selection, and final value placement while still treating examples as demonstrations rather than evidence for the current instance.

8. Postprocessing and Output Normalization

DRILLER uses deterministic postprocessing to align model outputs with the final schema interface. Even with a strict JSON Schema response format, outputs can contain representation artifacts such as literal Unicode escape strings or the string `"null"` where JSON `null` was intended. Postprocessing cleans these artifacts without changing the semantic content of the extraction.

Postprocessing begins with JSON parsing: if an output cannot be parsed as JSON, the trial is marked invalid rather than repaired by string heuristics. This boundary is especially important for `Mixed-SC`, where aggregation should combine valid candidate objects rather than fragments of malformed responses.

DRILLER then applies Unicode cleanup recursively to string values. Spanish text may include diacritics, the letter ñ, inverted punctuation marks, and other characters that can appear inconsistently across model outputs or API layers. The normalizer cleans literal escape artifacts such as `\u00e1` when they appear inside strings and then normalizes strings to Unicode Normalization Form C (NFC). NFC makes visually identical strings more stable for comparison while preserving their textual content.

The prompt asks the model to use JSON `null` for unsupported information when null is allowed, but models may emit string literals such as `"null"`, `"NULL"`, or other casing variants instead. DRILLER converts these string markers to JSON `null` only at schema positions where `null` is valid. It does not globally replace all occurrences, because a non-nullable string field should not be silently changed into a null value.

Each `Mixed-SC` trial independently performs prompt construction, strict generation, parsing, Unicode cleanup, schema restoration when relevant, and null coercion. Invalid trials are excluded from aggregation. The aggregator receives postprocessed JSON candidates in the original schema shape; after aggregation, nullable-string coercion may be applied again so selected values remain aligned with schema-valid null representation.

9. Heterogeneous Self-Consistency

9.1. Motivation

Schema-guided extraction can vary across generations even when the output format is constrained. Candidate outputs may differ in whether they include a borderline entity, how they normalize a date, which enum label they choose, or whether they return JSON `null`. Some variance comes from sampling, and some comes from the interface shown to the model. The reasoning-augmented prompt, for example, spends more output budget on field-local evidence assessment before committing to values.

`Mixed-SC` thus uses heterogeneous self-consistency [15]. Rather than sampling one prompt repeatedly, it combines `Schema-RAG`, which produces final values directly, with `Reasoned-RAG`, which temporarily produces `{reasoning, value}` objects and then unwraps them. This can expose different error patterns to the aggregator. The final decision is not a flat vote over complete JSON strings; it is a recursive schema-aware aggregation over candidate objects.

Figure 5 illustrates how `Mixed-SC` aggregates heterogeneous candidates recursively over the target schema.

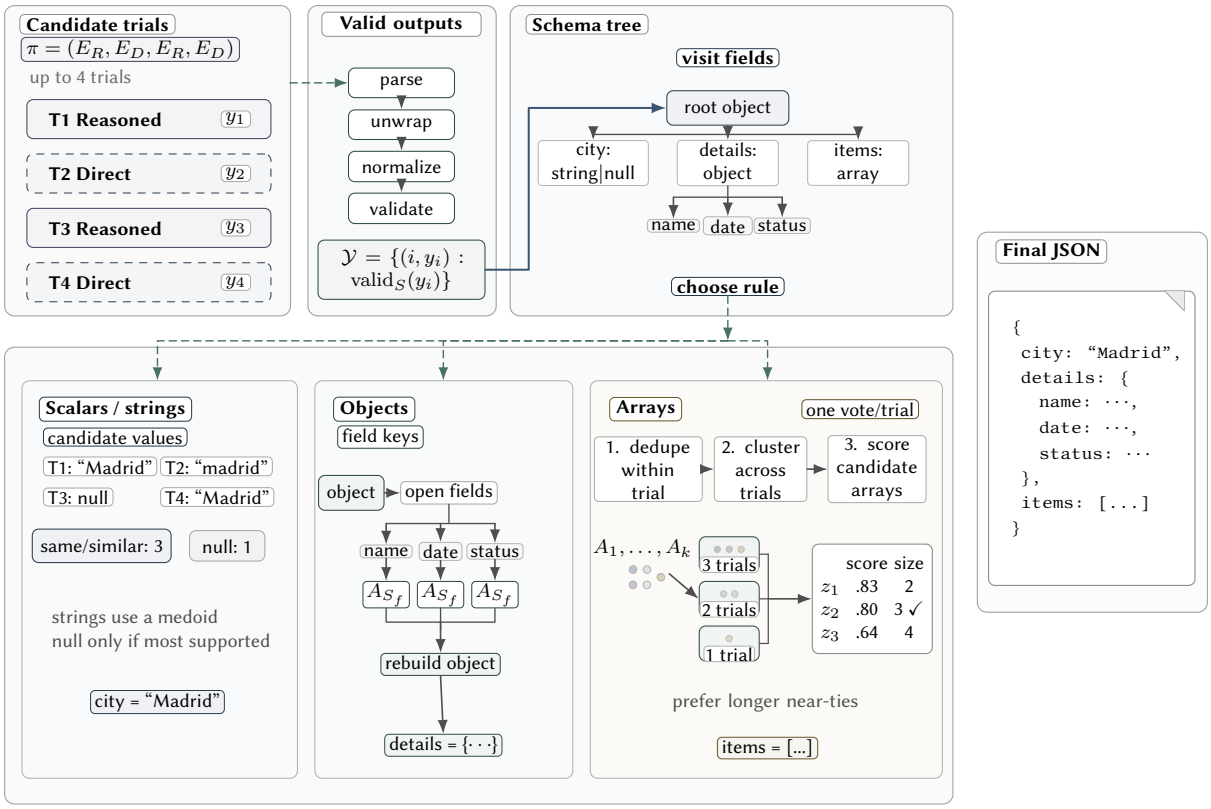

Figure 5: Schema-aware heterogeneous self-consistency in DRILLER. Mixed-SC executes a budget-aware prefix of interleaved reasoning-augmented and direct trials, postprocesses valid outputs into the original challenge schema, and aggregates them by schema position rather than by whole-JSON string. Scalar and string positions are selected by support-based clustering, object positions are rebuilt through recursive field-wise aggregation, and array positions use within-trial deduplication, cross-trial item clustering, and recall-biased MBR-style selection. The final JSON is reconstructed in the original schema shape.

9.2. Candidate Generation

Let $q = (x, S)$ denote an extraction instance with source text x and target schema S. Let E_D be the direct extractor used by `Schema-RAG`, and let E_R be the reasoning-augmented extractor used by `Reasoned-RAG`. `Mixed-SC` defines the intended heterogeneous plan

$$\pi = (E_R, E_D, E_R, E_D).$$

Budget-aware execution uses an interleaved order so that, if the runner stops early, the available candidate set should not consist only of one extractor family. Interleaving exposes the aggregator to both direct and reasoning-augmented candidates as early as possible, so partial execution still preserves the heterogeneous premise of `Mixed-SC`. The plan starts with the reasoning-augmented extractor to prioritize the more evidence-sensitive configuration.

If $k \leq 4$ trials fit under the effective token and time budgets, the runner executes the ordered prefix with completed trial indexes $K = \{1, \ldots, k\}$. For each $i \in K$, the extractor π_i performs its own RAG retrieval, prompt construction, strict JSON generation, and deterministic postprocessing. Direct trials render examples and schemas in the `Schema-RAG` format. Reasoning-augmented trials render examples and schemas in the temporary `Reasoned-RAG` format, so demonstrations contain reasoning/value fields during generation.

Postprocessing applies Unicode normalization and schema-aware null coercion to map each raw model response to either a final-shape candidate y_i or an invalid output, rejecting malformed outputs. The aggregation input is therefore

$$Y = \{(i, y_i) : i \in K,\ \mathrm{valid}_S(y_i)\}.$$

Each pair (i, y_i) preserves the trial identity used for support counting. The value y_i is always in the original challenge schema shape; raw model messages and reasoning traces are not aggregated. If $Y = \emptyset$, the system does not fabricate an extraction from partial information.

Before the first trial, the runner uses an incremental budget planner to estimate how many planned attempts fit under the effective token and time budgets for the instance. After each completed extraction, it updates the estimate using observed prompt tokens when available, completion tokens, and elapsed trial time, then decides whether another planned extraction can still be attempted. Because the system does not fully control hosted-call token use, this continuation decision is made after completed trials rather than by assuming that all four planned calls will fit.

9.3. Schema-Aware Recursive Aggregation

Let $A_S(Y)$ denote recursive aggregation of indexed candidate values Y under schema S. Complete JSON objects are not treated as indivisible. If S is an object schema with fields F_S, each emitted field is aggregated from the values observed for that field:

$$Y_f = \{(i, y_i[f]) : (i, y_i) \in Y,\ y_i \text{ is an object},\ f \in y_i\},$$

$$A_S(Y)[f] = A_{S_f}(Y_f),$$

where S_f is the child schema for field f. Required root fields are always considered when present in the schema. Optional nested fields are emitted only when enough candidates contain the property, so a single stray optional key does not automatically enter the final object. Field-wise recursion allows one candidate to contribute stronger support for one field while another contributes a better-supported value elsewhere.

Recursive aggregation compares values with a schema-aware similarity function $\mathrm{sim}_S(a, b) \in [0, 1]$. For exact scalar types, including enums, booleans, integers, and numeric values, agreement is canonical equality:

$$\mathrm{sim}_S(a, b) = \mathbb{1}[a = b].$$

String similarity uses the configured comparator. The submitted default normalizes case and accents, then combines token overlap and character n-gram overlap. Minor spelling, accent, punctuation, or phrasing differences can cluster together, but the aggregator still selects a generated string.

Object similarity is computed field by field. Let $F(a, b) \subseteq F_S$ be the fields present in at least one of the two objects, and let w_f be a small schema-dependent field weight. Missing-on-one-side fields contribute zero similarity but still count in the denominator:

$$\mathrm{sim}_S(a, b) = \frac{\sum_{f \in F(a,b)} w_f \mathbb{1}[f \in a \wedge f \in b]\, \mathrm{sim}_{S_f}(a[f], b[f])}{\sum_{f \in F(a,b)} w_f}.$$

Array similarity ignores exact position and compares items by a greedy one-to-one soft matching. If $M(a, b)$ is the greedy matching between items of arrays a and b, using the item-schema similarity $\mathrm{sim}_{S_{\mathrm{item}}}$, then

$$\mathrm{match}(a, b) = \sum_{(u,v) \in M(a,b)} \mathrm{sim}_{S_{\mathrm{item}}}(u, v),$$

$$\mathrm{sim}_S(a, b) = \frac{\mathrm{match}(a, b)}{|a| + |b| - \mathrm{match}(a, b)}.$$

Two empty arrays have similarity 1, while an empty and a non-empty array have similarity 0. This soft Jaccard-style score lets two arrays agree strongly when items appear in different orders or when object items differ only in secondary fields.

9.4. Clustering and Scalar Selection

At each scalar or object position, let

$$V = \{(i, v_i) : (i, y_i) \in Y\}$$

denote the indexed values observed there. `Mixed-SC` clusters exact types by canonical equality. For non-exact types, candidates are processed in stable order and assigned to the compatible cluster with highest similarity, subject to two constraints: the similarity must reach the active threshold τ_S, and the cluster may contain at most one value from any trial. A cluster's support is therefore trial support, not occurrence count:

$$\mathrm{supp}(C) = |\{i : (i, v) \in C\}|.$$

Clusters are ranked by decreasing support, with stable first occurrence as a tie-breaker. Let $C^\star$ be the highest-ranked non-null cluster. For non-exact clusters, the selected representative is a medoid: a generated candidate value whose average schema-aware similarity to the cluster is maximal,

$$\mathrm{medoid}_S(C) = \arg\max_{x \in C} \frac{1}{|C|} \sum_{y \in C} \mathrm{sim}_S(x, y).$$

The aggregator therefore chooses among generated values rather than composing a new free-text answer. For object clusters, the representative is obtained by recursively aggregating the object fields of the cluster members.

For null values, let $n_\varnothing$ be the number of candidate values equal to JSON `null`. For nullable fields, `null` is selected only when its support is strictly greater than the best non-null support:

$$A_S(V) = \texttt{null} \quad \Longleftrightarrow \quad n_\varnothing > \mathrm{supp}(C^\star).$$

This tie policy avoids turning uncertainty into absence while still preserving shared abstention when multiple trials independently return `null`. If all candidates are null, the nullable field is returned as `null`; for non-nullable schemas, the aggregator falls back to the conservative empty value for the schema type.

9.5. Array Aggregation

Array disagreement can involve order, omissions, duplicates, merged items, split items, or outliers, so majority voting over complete arrays would be brittle. Two candidates may contain the same substantive items in different orders or differ by one peripheral object. DRILLER therefore aggregates arrays by clustering items across candidate arrays.

For an array field, let A_i be the observed array from trial i. Items are first deduplicated within each A_i using the item-schema similarity and a high intra-trial threshold. This prevents repeated near-duplicates from one generated array from becoming multiple votes. The remaining indexed item candidates are then clustered across trials using $\mathrm{sim}_{S_{\mathrm{item}}}$, again with the restriction that each cluster can contain at most one item from a given trial.

To improve alignment, object-array clustering may use a field that behaves like an identifier: a name, title, mention, text span, ingredient, symptom, reaction, or source fragment. When such a field h is selected with sufficient confidence, item clustering uses similarity on h to decide whether two objects refer to the same extracted entity. The fields inside an aligned object cluster are aggregated recursively. If no useful identity field is detected, the aggregator falls back to broader object similarity.

Object arrays often combine extraction and classification, and the identity-like field tells the system which candidate objects correspond; remaining fields can then be compared or aggregated inside that item cluster. Imperfect identity detection is also a risk: a generic attribute may align unrelated objects, while failure to detect an identity field may split corresponding objects into duplicate clusters.

9.6. Recall-Biased Selection

After item clustering, the aggregator builds a small set $\mathcal{Z}$ of candidate final arrays from two construction strategies. One family uses support thresholds:

$$z_m = [\mathrm{rep}(C) : \mathrm{supp}(C) \geq m] \,,$$

where m ranges over configured support floors and support ratios, and $\mathrm{rep}(C)$ is the medoid or recursively aggregated representative of item cluster C. A second candidate is built greedily in a minimum Bayes risk (MBR) style [16]: starting from the empty array, the builder adds the cluster whose representative most improves agreement with the observed arrays, stopping when no remaining cluster improves the score.

For any candidate final array z, the MBR score is its average schema-aware agreement with the observed arrays:

$$R(z) = \frac{1}{|\mathcal{A}|} \sum_{A_i \in \mathcal{A}} \mathrm{sim}_{S_{\mathrm{array}}}(z, A_i),$$

where $\mathcal{A}$ is the set of non-null observed arrays for the field. Pure MBR would select the candidate with maximal $R(z)$. The submitted selector is recall-biased within a small tolerance. Let

$$R_{\max} = \max_{z \in \mathcal{Z}} R(z), \qquad \mathcal{E} = \{z \in \mathcal{Z} : R(z) \geq R_{\max} - \epsilon\}.$$

The final array is selected from $\mathcal{E}$ by preferring, in order, greater length, higher MBR score, and earlier candidate order:

$$z^\star = \arg\max_{z \in \mathcal{E}} \left(|z|,\ R(z),\ -\mathrm{order}(z)\right).$$

The default tolerance is $\epsilon = 0.06$, configurable at runtime. This favors retaining supported items when evidence from trials is close. The longer array is not accepted unconditionally; it must remain competitive under the consensus objective.

The recall-biased selector trades omission against over-extraction. A stricter selector would discard more borderline items, improving precision in some cases but risking missed entities or list elements. DRILLER keeps items with enough support to remain near the best consensus score, which is useful for unordered lists, object arrays, and evidence collections where candidates often agree on a core set but vary on peripheral supported items.

9.7. Conservativeness and Failure Modes

Conservative aggregation selects among generated candidates rather than inventing semantic content. Exact scalar types are selected by canonical equality groups; non-exact string clusters choose a generated representative; arrays are built from generated item clusters. The aggregator does not use external knowledge, synthesize new enum labels, or create facts absent from all candidate outputs.

Self-consistency still does not guarantee correctness, since multiple trials can share the same misunderstanding of a field description, attend to the same distractor mention, or overtrust an unsupported fact; aggregation can preserve that mistake with high support. Heterogeneous prompting reduces reliance on one prompt representation, but all trials still share the source text, target schema, local FSP corpus, model endpoint, and broad extraction rules.

For arrays, the recall-biased selector may retain extra borderline items, and identity-field detection can misalign object items or fail to merge true matches. Threshold choices also matter: loose thresholds can merge distinct values, while strict thresholds can split legitimate paraphrases or duplicate entities. These risks are the cost of trying to preserve recall in schemas where missing list members are common and where item order is not meaningful.

`Mixed-SC` increases inference cost by requesting up to four model calls rather than one, with up to two reasoning-augmented calls that can produce longer intermediate outputs before unwrapping.

10. Experimental Setup

The submitted solution consists of three final pipelines: `Schema-RAG`, `Reasoned-RAG`, and `Mixed-SC`. These correspond to the submitted names `enriched-schema-rag`, `enriched-inline-reasoning-rag`, and `mixed-extractors-self-consistency-rag`. All three use the retrieval and schema-rendering mechanisms described above.

We report the organizer-provided official GenSIE results. The official scoring set contains 125 instances: the released 145-instance gold test set minus a uniform 20-instance drop-set. The excluded groups are 8 `medical_trials`, 2 `cultural_monuments`, and 10 `stem_biology` instances, removed because their schemas could not be scored reliably in the constrained-decoding inference stack. The exclusion is applied uniformly to all systems and to both baselines [2].

The official leaderboard uses micro-averaged precision, recall, and F_1 over flattened schema leaves. It ranks systems by the average model-specific gap closed,

$$\max\left(0, \frac{F_1^{system} - F_1^{baseline}}{1 - F_1^{baseline}}\right),$$

using fixed no-think baselines: $F_1 = 0.7895$ for Gemma 4 E4B and $F_1 = 0.7805$ for Qwen3-14B. For each model, the leaderboard takes the best reported cell for a system, regardless of whether that cell was run in `think` or `no-think` mode, but always computes gap closed against these no-think baselines.

We do not report wall-clock-time comparisons. The official reports do not isolate our code from hosted backend latency and small-language-model generation time. Token counts are also omitted from the main tables because the ranking is quality-based and the official summary metrics are computed after the 20-instance filter.

After the drop-set, the scored instances contain 20 unique structural schema fingerprints under the same normalization used by retrieval. Nine fingerprints are present in the development/FSP corpus and account for 34 scored instances; 11 are unseen and account for 91 scored instances. The retrieval replay routes the 34 seen-fingerprint instances to same-schema retrieval and the remaining 91 to field-aware fallback. No retrieval-failure route remains after the official exclusions.

11. Results and Analysis

DRILLER ranks first in the official leaderboard, with average gap closed 0.2158, reported as 21.6%. Table 5 lists all official DRILLER cells after deduplicating repeated source events for the same model, pipeline, and mode. The best Gemma cell is `Mixed-SC` in `think` mode ($F_1 = 0.8285$, 18.5% gap closed). The best Qwen cell is `Schema-RAG` in `no-think` mode ($F_1 = 0.8346$, 24.6% gap closed). These two cells are the pair used for the team-level average.

11.1. Pipeline Comparisons

Subsequent comparisons use the best cell for each model and pipeline, following the leaderboard policy. Table 6 shows these cells. `Schema-RAG` is the strongest single-call pipeline for both models when each pipeline may use its best mode. Relative to the official no-think baselines, best-cell `Schema-RAG` improves Gemma by $+0.0361$ F_1 and Qwen by $+0.0541$ F_1, closing 17.1% and 24.6% of the respective

Table 5
Official DRILLER cells on the 125 scored GenSIE instances. Gap is computed against fixed no-think baselines: Gemma $F_1 = 0.7895$, Qwen $F_1 = 0.7805$.

Model	Pipeline	Mode	Prec.	Rec.	F_1	Gap
Gemma	`Mixed-SC`	think	.8285	.8285	.8285	18.5%
Gemma	`Schema-RAG`	think	.8256	.8256	.8256	17.1%
Gemma	`Schema-RAG`	no-think	.8211	.8211	.8211	15.0%
Gemma	`Reasoned-RAG`	think	.8249	.8115	.8181	13.6%
Gemma	`Reasoned-RAG`	no-think	.8239	.8113	.8176	13.3%
Gemma	`Mixed-SC`	no-think	.8241	.8090	.8165	12.8%
Qwen	`Schema-RAG`	no-think	.8346	.8346	.8346	24.6%
Qwen	`Reasoned-RAG`	think	.8297	.8297	.8297	22.4%
Qwen	`Reasoned-RAG`	no-think	.8280	.8280	.8280	21.6%
Qwen	`Schema-RAG`	think	.8201	.8201	.8201	18.0%
Qwen	`Mixed-SC`	think	.8177	.8177	.8177	16.9%
Qwen	`Mixed-SC`	no-think	.8129	.8129	.8129	14.8%

baseline gaps. Official per-route baseline reports are not present in the released DRILLER artifacts, so route-level diagnostics below are system-only cuts rather than route-specific baseline deltas.

Table 6
Best official cell by model and submitted pipeline.

Model	Pipeline	Mode	Prec.	Rec.	F_1	Gap
Gemma	`Schema-RAG`	think	.8256	.8256	.8256	17.1%
Gemma	`Reasoned-RAG`	think	.8249	.8115	.8181	13.6%
Gemma	`Mixed-SC`	think	.8285	.8285	.8285	18.5%
Qwen	`Schema-RAG`	no-think	.8346	.8346	.8346	24.6%
Qwen	`Reasoned-RAG`	think	.8297	.8297	.8297	22.4%
Qwen	`Mixed-SC`	think	.8177	.8177	.8177	16.9%

Table 7 isolates the aggregate comparison between `Schema-RAG` and the official no-think baseline for each model. This is not a controlled ablation of retrieval alone, because `Schema-RAG` also changes schema rendering and prompt format relative to the organizer baseline. It is nevertheless the clean official comparison between the submitted single-call RAG extractor and the fixed no-think floor used by the leaderboard.

Table 7
`Schema-RAG` compared with official no-think baselines on the 125 scored instances.

Model	`Schema-RAG` mode	Baseline F_1	System F_1	ΔF_1	Gap
Gemma	think	.7895	.8256	+.0361	17.1%
Qwen	no-think	.7805	.8346	+.0541	24.6%

11.2. Retrieval Routes

Table 8 reports best-cell performance by retrieval route. Same-schema retrieval covers 34 scored instances, while Field-RAG covers 91. The Field-RAG subset is not a weaker residual subset in the official scores: for every best-cell pipeline, its F_1 is higher than the same-schema subset. This should not be read as a causal advantage of fallback retrieval. The two routes correspond to different schema families after the official drop-set, and the excluded schemas remove the previous retrieval-failure path.

11.3. Reasoning and Self-Consistency

Following the best-cell comparison in Table 6, the temporary reasoning/value interface is an inference-time variant of the shared RAG extractor, but the official cells do not show it overtaking direct

Table 8
Best-cell performance by retrieval route after the official drop-set. Same-schema contains 34 scored instances; Field-RAG contains 91 scored instances.

		Same-schema			Field-RAG		
Model	Pipeline	Prec.	Rec.	F_1	Prec.	Rec.	F_1
Gemma	`Schema-RAG`	.7770	.7770	.7770	.8429	.8429	.8429
Gemma	`Reasoned-RAG`	.8074	.8074	.8074	.8312	.8129	.8220
Gemma	`Mixed-SC`	.8067	.8067	.8067	.8363	.8363	.8363
Qwen	`Schema-RAG`	.7819	.7819	.7819	.8534	.8534	.8534
Qwen	`Reasoned-RAG`	.7836	.7836	.7836	.8461	.8461	.8461
Qwen	`Mixed-SC`	.7795	.7795	.7795	.8314	.8314	.8314

`Schema-RAG`. Best-cell `Reasoned-RAG` reaches $F_1 = 0.8181$ with Gemma and $F_1 = 0.8297$ with Qwen, below `Schema-RAG` in both models. The no-think `Reasoned-RAG` cells are also below the corresponding `Schema-RAG` no-think cells for the models where both are available.

`Mixed-SC` is the best official Gemma cell because the `think` `Mixed-SC` run edges ahead of Gemma `Schema-RAG` by $+0.0029$ F_1. This does not support a broad claim that self-consistency is robustly beneficial. With Qwen, best-cell `Mixed-SC` is lower than best-cell `Schema-RAG` by -0.0169 F_1. In like-for-like no-think comparisons, `Mixed-SC` is below `Schema-RAG` for both models: .8165 versus .8211 with Gemma, and .8129 versus .8346 with Qwen. The extra compute and blind resampling in `Mixed-SC` therefore appear useful for one Gemma thinking cell, but not as a consistent default.

11.4. Failure Patterns

The main official scoring caveat is the 20-instance drop-set [2]. It removes two seen-schema `MonumentBIC` instances and 18 unseen-schema instances from `ClinicalTrialRecord` and `TaxonomicClassification`. After this filter, the retrieval replay has no failed retrieval routes, and all 91 fallback instances have field-level pair diagnostics. The remaining errors are semantic and structural extraction errors inside scored instances rather than unscoreable schema-stack failures.

We keep per-domain tables, selected-example hub counts, and retrieval-score distributions in Appendix A. These diagnostics are useful for interpreting the submitted system, but they are not controlled causal ablations.

12. Limitations

Several limitations follow from DRILLER's schema-variable design. The first is dependence on a local few-shot prompting corpus. The demonstrations are synthetic and curated, which gives controlled coverage of nulls, populated values, enum choices, arrays, nested structures, and distractors, but also requires careful construction and review. Each case must keep the instruction, source text, schema, output, and rationale mutually consistent. Errors or narrow coverage in the corpus can propagate into prompts.

Coverage remains incomplete because GenSIE schemas vary by instance. A demonstration family can represent one schema well while failing to cover later fields with different names, descriptions, nesting, or value regimes. Complementary examples reduce reliance on one output pattern, but they cannot guarantee that every private evaluation schema has a close analogue. If retrieved examples mostly show populated arrays, frequent enum labels, or non-null values, they may still bias the model toward overproduction.

Retrieval depends on schema information quality. Same-schema retrieval is useful when structurally matching examples exist, but it has limited value when the local corpus lacks the target schema. Field-level fallback is more flexible, yet it relies on names, descriptions, type representations, and coarse type classes. Sparse or generic descriptions can make unrelated fields appear similar, while equivalent fields

with different wording can be missed. Type compatibility prevents many implausible alignments, but it can also exclude examples whose meanings are related despite different schema forms.

The reasoning/value wrapper in `Reasoned-RAG` is a cost trade-off. It can encourage explicit evidence assessment and clearer decisions between a value and `null`, but it increases output length, consumes context, and may raise latency or token cost. Because the submitted wrapper is top-level only, it does not give every nested field its own explicit evidence scaffold. This keeps the temporary schema manageable, but complex nested objects may still require decisions that are only indirectly guided by top-level reasoning.

`Mixed-SC` further increases inference cost by requesting up to four trials, including up to two reasoning-augmented calls. Runtime budgets may limit completed trials, and practical efficiency depends on the model, backend, decoding configuration, and official token accounting. Its value must be judged jointly by extraction quality and cost.

Aggregation is heuristic. It votes over closed scalar values, clusters strings, aggregates objects field by field, clusters array items, and selects arrays with a recall-biased strategy. These rules are schema-aware and conservative in the sense that selected representatives come from generated candidates, but they are not optimality guarantees. Shared mistakes can be reinforced, borderline array items can be retained, and imperfect identity-field detection can misalign object arrays.

Component-level effects remain empirical questions. Claims about the separate impact of schema rendering, retrieval, same-schema prompting, reasoning wrappers, postprocessing, or heterogeneous self-consistency require controlled ablations. Results may also depend strongly on the hosted model's structured-output behavior, context limits, multilingual extraction ability, and instruction following.

13. Conclusion

DRILLER submitted a modular schema-guided extraction system for GenSIE 2026. It treats each instance as information extraction in Spanish under a variable JSON Schema, where the central challenge is interpreting field semantics from the current instruction, schema, and source text while returning grounded schema-conformant JSON.

The three submitted pipelines are `Schema-RAG`, `Reasoned-RAG`, and `Mixed-SC`. All use retrieval-augmented prompting. The common extraction spine combines schema-readable Pydantic-style prompting, strict structured JSON generation, field-aware few-shot retrieval, and conservative postprocessing. `Reasoned-RAG` adds temporary top-level reasoning/value wrappers that are removed before final output. `Mixed-SC` combines direct and reasoning-augmented candidates through heterogeneous self-consistency and recursive schema-aware aggregation.

The main design principle is to treat the schema as a runtime semantic specification rather than only an output validator. Readable prompt schemas help the model interpret fields; strict generation constrains the response interface; retrieved examples illustrate field behavior; postprocessing normalizes representation-level artifacts without semantic repair; and aggregation reconciles candidate values according to schema type.

Future work should strengthen retrieval and aggregation. Learned retrieval could complement schema fingerprints and field-similarity heuristics, especially when field descriptions are sparse. Automatic validation of FSP cases would help detect inconsistent demonstrations before they enter prompts. Aggregation could benefit from better uncertainty estimates, learned item alignment, and schema-specific constraints. Controlled ablations are needed to measure component contributions, and robust array/object extraction remains a priority because omissions, duplicates, and field-alignment errors interact strongly in nested structures.

Declaration on Generative AI

During the preparation of this manuscript, generative AI tools assisted with repository inspection, organization of technical notes, drafting, and language editing. The authors reviewed, edited, and

approved the final content and take full responsibility for the submitted paper. DRILLER also uses synthetic and curated few-shot resources as part of its retrieval component; an agentic LLM coding assistant helped prepare provisional resources, and the authors manually reviewed and validated them before use.

A. Additional Diagnostic Analyses

This appendix reports retrieval and schema-composition diagnostics after applying the official 20-instance drop-set. These diagnostics characterize what the submitted RAG pipelines receive on the 125 scored instances; they are not controlled ablations.

A.1. Schema and Retrieval Composition

Table 9
Schema-composition diagnostics over the 125 scored instances.

Quantity	Value
Unique schema fingerprints	20
Schema fingerprints seen in development/FSP corpus	9 (45.0%)
Schema fingerprints unseen in development/FSP corpus	11 (55.0%)
Scored instances with seen schema fingerprints	34 (27.2%)
Scored instances with unseen schema fingerprints	91 (72.8%)
Instances routed to same-schema retrieval	34
Instances routed to field-aware fallback	91
Field-aware instances with pair diagnostics	91
Field-aware instances on retrieval-failure path	0
Complementary pair differs from independent top-2	91/91
Complementary pair improves field coverage	40/91

Table 10
Scored schemas by development-set overlap after the official drop-set.

Seen	Schema title	Tasks	Domains
Yes	CelestialObjectProperties	5	stem
Yes	ContractSummary	3	legal
Yes	DiseaseProfile	6	medical
Yes	LiteraryWork	5	cultural
Yes	MediaReview	3	cultural
Yes	NamedEntities	1	cultural
Yes	NewsArticle	7	general
Yes	SoftwareDescription	2	technical
Yes	TextAnswer	2	cultural
No	AlbumReview	10	cultural
No	ArtistProfile	5	cultural
No	CourtCaseTimeline	10	legal
No	DischargeSummary	10	medical
No	EnvironmentalRuling	7	environmental
No	LegalInquiry	6	legal
No	ObservationLog	10	stem
No	PaperArgument	10	research
No	SecurityIncident	10	technical
No	SentencingLogic	3	legal
No	TheaterPlaybill	10	cultural

The same-schema route is stable: all 34 structurally seen scored cases are routed through it, with mean selected-pair minimum similarity of .957. For the 91 field-aware cases, the complementary pair differs from the independent top-2 in every case. The fallback therefore trades individual nearest-neighbor score for pair complementarity; coverage improves in 40 of the 91 field-aware instances.

Table 11
Retrieval route by scored test domain.

Domain	Tasks	Same	Field	Failed
cultural_album_review	10	0	10	0
cultural_entities	1	1	0	0
cultural_extraction	2	2	0	0
cultural_literature	5	5	0	0
cultural_media	3	3	0	0
cultural_people	5	0	5	0
cultural_theater	10	0	10	0
environmental_assessments	7	0	7	0
general_disasters	7	7	0	0
legal_case_timeline	10	0	10	0
legal_contracts	3	3	0	0
legal_judicial	3	0	3	0
legal_legislation	6	0	6	0
medical_discharge	10	0	10	0
medical_diseases	6	6	0	0
research_paper_argument	10	0	10	0
stem_astronomy	2	2	0	0
stem_astronomy_detailed	3	3	0	0
stem_observations	10	0	10	0
technical_security_incident	10	0	10	0
technical_software	2	2	0	0

Table 12
Summary statistics for same-schema and field-aware retrieval decisions on scored instances.

Diagnostic	Min	P25	Mean	Med.	P75	Max
Selected same-schema min similarity	.861	.950	.957	.971	.981	.993
Same-schema candidate max similarity	.877	.955	.964	.974	.989	.999
Field pair-score gain vs. independent top-2	.514	.660	.786	.687	.920	1.104
Field-score sum loss vs. independent top-2	.035	.138	.241	.246	.349	.527
Field coverage gain vs. independent top-2	.000	.000	.063	.000	.111	.250

Table 13
Most frequently selected FSP examples in field-aware retrieval after the official drop-set. Counts are over the 91 field-aware scored instances.

FSP example	Selections
medical diseases: streptococcal pharyngitis	46
cultural media: mixed series	23
medical drug: ibuprofen	23
cultural literature: city of mirrors	20
environmental ecology: river spill	20
stem astronomy: Mars	20
legal contracts: agricultural machinery sale	16
cultural monuments: solana shelters	13
legal legislation: green patios	13
lifestyle recipes: eggplants with tahini	10

References

[1] Z. Wen, S. Liang, Y. Wu, Y. Zhang, Y. Liu, Effective and efficient schema-aware information extraction using on-device large language models, 2025. URL: https://arxiv.org/abs/2505.14992. arXiv:2505.14992.

[2] A. Piad-Morffis, et al., Overview of GenSIE at IberLEF 2026: Schema-Guided Information Extraction with Small Language Models in Spanish, in: Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2026), co-located with the 42nd Conference of the Spanish Society for Natural Language Processing (SEPLN 2026), CEUR-WS.org, 2026.

[3] A. Bonet-Jover, J. Á. González-Barba, L. Chiruzzo, Overview of IberLEF 2026: Natural Language Processing Challenges for Spanish and other Iberian Languages, in: Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2026), co-located with the 42nd Conference of the Spanish Society for Natural Language Processing (SEPLN 2026), CEUR-WS.org, 2026.

[4] Z. Li, Y. Zeng, Y. Zuo, W. Ren, W. Liu, M. Su, Y. Guo, Y. Liu, X. Li, Z. Hu, L. Bai, W. Li, Y. Liu, P. Yang,

X. Jin, J. Guo, X. Cheng, KnowCoder: Coding structured knowledge into LLMs for universal information extraction, 2024. URL: https://arxiv.org/abs/2403.07969. arXiv:2403.07969.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: Advances in Neural Information Processing Systems, volume 33, 2020, pp. 1877–1901.

[6] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, W. Chen, What makes good in-context examples for GPT-3?, in: Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, Association for Computational Linguistics, Dublin, Ireland and Online, 2022, pp. 100–114. URL: https://aclanthology.org/2022.deelio-1.10/. doi:10.18653/v1/2022.deelio-1.10.

[7] O. Rubin, J. Herzig, J. Berant, Learning to retrieve prompts for in-context learning, in: Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Seattle, United States, 2022, pp. 2655–2671. URL: https://aclanthology.org/2022.naacl-main.191/. doi:10.18653/v1/2022.naacl-main.191.

[8] N. Reimers, I. Gurevych, Sentence-BERT: Sentence embeddings using Siamese BERT-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 3982–3992. URL: https://aclanthology.org/D19-1410/. doi:10.18653/v1/D19-1410.

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Advances in Neural Information Processing Systems, volume 33, 2020, pp. 9459–9474.

[10] Y. Yao, S. Mao, N. Zhang, X. Chen, S. Deng, X. Chen, H. Chen, Schema-aware reference as prompt improves data-efficient knowledge graph construction, 2023. URL: https://arxiv.org/abs/2210.10709. arXiv:2210.10709.

[11] S. Geng, M. Josifoski, M. Peyrard, R. West, Grammar-constrained decoding for structured NLP tasks without finetuning, 2023. URL: https://arxiv.org/abs/2305.13971. arXiv:2305.13971.

[12] S. Geng, H. Cooper, M. Moskal, S. Jenkins, J. Berman, N. Ranchin, R. West, E. Horvitz, H. Nori, Generating structured outputs from language models: Benchmark and studies, 2025. URL: https://arxiv.org/abs/2501.10868. arXiv:2501.10868.

[13] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Advances in Neural Information Processing Systems, volume 35, 2022, pp. 24824–24837.

[14] Y. Li, R. Ramprasad, C. Zhang, A simple but effective approach to improve structured language model output for information extraction, 2024. URL: https://arxiv.org/abs/2402.13364. arXiv:2402.13364.

[15] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, 2022. URL: https://arxiv.org/abs/2203.11171. arXiv:2203.11171.

[16] S. Kumar, W. Byrne, Minimum Bayes-risk decoding for statistical machine translation, in: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004, Association for Computational Linguistics, Boston, Massachusetts, USA, 2004, pp. 169–176.

[17] GenSIE Organizers, GenSIE 2026 submission guidelines, 2026. Challenge documentation.

[18] GenSIE Organizers, GenSIE 2026 task description, 2026. Challenge documentation.

[19] B. T. Willard, R. Louf, Efficient guided generation for large language models, 2023. URL: https://arxiv.org/abs/2307.09702. arXiv:2307.09702.