SEsml: Schema-guided Extraction with Small Language Models

Alejandro Beltrán[1]

[1]Universidad de La Habana, Havana, Cuba

Abstract

This paper presents **SEsml** (Schema-guided Extraction with Small Language Models), a particular solution submitted by a researcher at the Universidad de La Habana to the GenSIE 2026 shared task at IberLEF 2026. The task requires zero-shot extraction of nested JSON objects from Spanish texts using SLMs under 14B parameters. SEsml instantiates a central design principle — *decoupling semantic reasoning from structural enforcement* — as its defining architectural choice. The system argues that constrained decoding, while guaranteeing syntactic validity, imposes a measurable *constraint tax* on small models that degrades semantic accuracy. To mitigate this, the principal pipeline, *extraction*, employs a two-phase draft-then-decode architecture that makes the decoupling explicit: an unconstrained DraftEngine produces a semi-structured `campo: valor` representation, followed by a Constrained Decoder that maps the draft into schema-valid JSON under `response_format=json_schema`. Two lighter variants explore alternative trade-offs: *hybrid_cot* (single-call with visible chain-of-thought) and *adaptive* (task-adaptive prompt assembly via semantic similarity). All pipelines share a SchemaOptimizer that resolves $ref references, enforces structural constraints, and normalises schemas for constrained decoding engines. The paper documents the prompt evolution from a verbose v1 (~3,180 chars) to a compact v3b (~1,163 chars), finding that shorter, rule-based prompts consistently outperform longer instructions for SLMs. On the GenSIE 2026 development set (149 tasks), all three SEsml pipelines outperform the unconstrained baseline on both Gemma 4 E4B and Qwen3-14B, closing 13.4–43.6% of the gap to a perfect system. The extraction pipeline is most effective on the smaller Gemma model (gap-closed 0.1761), while the adaptive pipeline leads on Qwen3-14B (gap-closed 0.4363). The SEsml team placed 5th of 11 teams on the official leaderboard (mean GapClosed = 0.0481).

Keywords

structured extraction, small language models, constrained decoding, schema-guided generation, Spanish NLP, draft-then-decode

1. Introduction

The extraction of structured information from unstructured text is a fundamental task in natural language processing, underpinning applications from knowledge base construction to document understanding. While large language models (LLMs) have demonstrated remarkable capabilities in open-ended text generation, reliably extracting structured data — particularly in formats such as JSON — remains challenging, especially when using small language models (SLMs) that are practical for on-device or resource-constrained deployments.

The GenSIE (General-purpose Schema-guided Information Extraction) shared task, hosted at IberLEF 2026 [1], addresses this challenge head-on [2]. Participants must build systems that take a Spanish text fragment and an arbitrary JSON Schema, and produce a valid JSON object whose values are faithfully grounded in the input. The task is zero-shot: test schemas may be entirely unseen during development. All systems must use SLMs with fewer than 14 billion parameters, reflecting the realistic constraint that such models must be deployable on consumer hardware.

This paper presents **SEsml** (Schema-guided Extraction with Small Language Models), a particular solution submitted by the ExtractUH team of the Universidad de La Habana to the GenSIE 2026 shared task. SEsml comprises three complementary pipelines — *extraction*, *hybrid_cot*, and *adaptive* — all of which instantiate the same defining architectural choice: *decoupling semantic reasoning from structural enforcement.*

IberLEF 2026, September 2026, León, Spain
alejandro.beltran@rect.uh.cu (A. Beltrán)
0009-0009-8073-1033 (A. Beltrán)

The motivation for this principle arises from a well-documented limitation of constrained decoding. While techniques such as token masking and grammar-guided generation guarantee syntactically valid output, they impose a measurable *constraint tax* [3] on small models. When forced to simultaneously reason about content and maintain rigid JSON syntax, SLMs suffer from *capacity contention*: their limited representational capacity is split between semantic accuracy and syntactic compliance, degrading both. Recent work on *Draft-Conditioned Constrained Decoding* [4] and *In-Writing* [5] demonstrates that generating an unconstrained intermediate representation before applying structural constraints can substantially mitigate this degradation.

The principal pipeline, *extraction*, operationalises this insight through a two-phase architecture that makes the decoupling explicit: an unconstrained DraftEngine produces a semi-structured `campo: valor` representation without JSON syntax, followed by a Constrained Decoder that maps the draft into schema-valid JSON using `response_format=json_schema`. Two lighter variants — *hybrid_cot* (single-call with visible chain-of-thought) and *adaptive* (task-adaptive prompt assembly via semantic similarity) — explore alternative trade-offs between decoupling, latency, and schema complexity.

The pipelines are evaluated on the GenSIE development set (149 tasks) and the official test set (145 tasks) across two SLM families (Gemma 4 E4B, Qwen3-14B). On the development set, all three SEsml pipelines outperform the unconstrained baseline on both models. The extraction pipeline closes 17.6% of the gap to perfect on Gemma 4 E4B, while the adaptive pipeline achieves the best overall score on Qwen3-14B (gap-closed 0.4363). On the official leaderboard, the SEsml team placed 5th of 11 systems with a mean GapClosed of 0.0481.

The paper is organised as follows. Section 2 provides background on constrained decoding limitations and the draft-then-decode solution. Section 3 presents the overall system architecture. Section 4 details the schema optimisation approach. Section 5 describes the principal extraction pipeline. Section 6 covers the alternative pipelines. Section 7 discusses prompt engineering and the evolution of the prompt design. Section 8 presents experimental results, Section 9 analyses error patterns, and Section 10 concludes.

2. Background: The Challenge of Constrained Decoding for Small Language Models

2.1. Problem Setup

The GenSIE shared task requires systems to perform zero-shot extraction of nested JSON objects from general-domain Spanish texts, using language models with fewer than 14 billion parameters. For each instance, the system receives a text fragment, a natural-language instruction, and an arbitrary target schema defined as a JSON Schema (OpenAPI 3.0 subset). The output must be a valid JSON object that strictly conforms to the schema while containing only information faithfully extracted from the input text — returning `null` for any field whose value is not present.

This setting imposes three simultaneous demands on the model: (a) understand and reason about the Spanish input text, which may require resolving coreferences, handling rich morphology, and performing multi-hop inference across paragraphs; (b) interpret the target schema, which may include nested objects, enumerated types with strict case-sensitive matching, inter-field constraints, and hallucination traps where fields request plausible but absent information; and (c) produce syntactically perfect JSON, where a single misplaced comma, unescaped quote, or type mismatch renders the output unusable.

For small language models (SLMs) with limited representational capacity, these demands compete for the same finite cognitive budget — what the authors term *capacity contention*.

2.2. Why Constrained Decoding: A Task-Driven Necessity

The GenSIE task specification imposes a hard requirement: every output must be a syntactically valid JSON object conforming to the target schema. An output whose JSON cannot be parsed — due to a misplaced comma or unescaped quote — yields a structural failure that the scoring metric cannot evaluate, effectively scoring zero for that instance. While the Flattened Schema Scoring does credit

partially valid structures through its key-wise evaluation, a single unparseable output loses an entire instance's contribution to the final F1. For a shared task where every percentage point matters, unreliable syntax is a liability no competitive system can afford.

This is not merely a stylistic preference. Castillo [3] reports that across diverse extraction tasks, unconstrained SLMs (no `response_format` enforcement) produce syntactically valid JSON on only 60–70% of instances. The remaining 30–40% of outputs — containing trailing commas, unquoted keys, truncated objects, or extraneous text — are structurally useless regardless of their semantic content. For a system competing on F1, syntactic failure is catastrophic: it nullifies any semantic accuracy the model may have achieved.

Constrained decoding (CD) addresses this failure mode directly. At each generation step, CD intervenes at the token level, masking any continuation that would violate the target grammar or schema and renormalising over the remaining valid set [6, 7]. Tools such as Outlines, XGrammar [8], and OpenAI's `response_format=json_schema` compile JSON schemas into finite-state machines that enforce structural correctness token by token. The result is predictable and well-documented: schema validity rises from the unconstrained baseline of 60–70% to effectively 100%, eliminating syntax errors, missing fields, and extraneous commentary [3]. Modern engines achieve this with minimal latency overhead — XGrammar reports near-zero added cost per token [8], and coalescence techniques can even accelerate generation by up to 50% over unconstrained decoding by skipping deterministic scaffolding [9, 10].

These properties make CD one of the most viable options for the GenSIE task: no system that accepts 30–40% structural failure rates can be competitive in a shared task where every percentage point of F1 matters. SEsml therefore adopts constrained decoding as its structural backbone — but with full awareness that CD is not a free lunch. The same mechanism that guarantees syntactic validity also imposes a measurable *constraint tax* on semantic accuracy, particularly for small models. The following subsections analyse this tax and present the mitigation strategies that are the paper's central contribution.

2.3. The Constraint Tax

However, CD is not a passive formatting filter. Because it intervenes at every generation step, it alters the model's internal probability distribution in ways that can degrade semantic accuracy. This phenomenon, termed the *constraint tax*, is particularly severe for small models.

Reddy et al. [4] formalise the mechanism through a KL-divergence lens: at each step, CD projects the model's original distribution onto the set of valid continuations via reverse-KL minimisation. When the model assigns low probability to valid tokens — which is typical when generating rigid syntax tokens such as braces, commas, or field names — this projection constitutes a large perturbation. Repeating this perturbation over hundreds of steps induces a cumulative *projection tax* that systematically biases decoding toward locally valid but semantically incorrect trajectories.

The empirical magnitude of the constraint tax is striking. Castillo [3] conducted a controlled 15,000-generation study comparing unconstrained prompting with hard schema-constrained decoding on SLMs. Hard CD raised schema validity from 61.5% to 100.0%, but simultaneously lowered answer accuracy from 19.7% to 11.0%. The proportion of outputs that were structurally valid but semantically wrong rose from 49.5% to 88.9% — the model produced *more* wrong answers precisely because it was forced into valid JSON. Schall and De Melo [11] independently confirmed this effect across diverse model architectures and task types, showing that constraint-induced performance degradation is inversely correlated with model scale: smaller models suffer disproportionately larger accuracy drops under identical constraints.

2.4. Capacity Contention and Structure Snowballing

The constraint tax manifests through two interrelated failure modes directly observed during system development.

Capacity contention occurs when an SLM attempts to reason about semantic content while simultaneously maintaining rigid JSON syntax. The model's limited representational capacity forces a trade-off: attention devoted to tracking open braces, comma placement, and quote escaping is attention withdrawn from entity recognition, relationship extraction, and value verification. This is not merely a prompting issue — it is a fundamental consequence of autoregressive generation under conflicting objectives.

Structure snowballing, identified by Zhou et al. [12], represents an even more insidious failure. Under Outlines-based constrained decoding, the cognitive load of formatting compliance can push the model into *formatting traps* — repetitive, self-reinforcing cycles of syntactic correction that consume generation capacity without resolving the underlying semantic errors. The model achieves perfect superficial syntactic alignment while entirely missing deep semantic errors. Zhou et al. term this the *alignment tax* of constrained decoding: a tension between structural granularity and internal model capacity.

Nguyen et al. [5] demonstrated that this fragility extends beyond grammar-level constraints to simple lexical ones. Instruction-tuned models suffered 14–48% comprehensiveness loss when asked to avoid a single common word or punctuation character. Through mechanistic analysis, they identified this as a *planning failure*: the instruction-tuned model's response template was disrupted by the constraint, causing it to abandon its normal reasoning process. Notably, base (non-instruction-tuned) models showed no systematic collapse, confirming that instruction tuning itself creates the coupling between task competence and narrow surface-form templates.

2.5. Draft-then-Decode: Decoupling Reasoning from Formatting

A growing body of work demonstrates that these failure modes can be mitigated by *decoupling* the reasoning phase from the formatting phase. The key insight is that if a model can express its understanding in a natural, unconstrained form before being required to produce structured output, the syntactic constraints no longer compete for the same cognitive budget.

Reddy et al. [4] propose *Draft-Conditioned Constrained Decoding* (DCCD), a training-free two-stage procedure. In the first stage, a model generates an unconstrained draft — free text, reasoning steps, or semi-structured notes — capturing the semantic plan. In the second stage, constrained decoding is applied *conditioned on this draft*, so that the model primarily performs structured realisation rather than open-ended reasoning. The draft shifts probability mass toward schema-consistent continuations, substantially reducing the projection tax. Across GSM8K, MATH500, and FOLIO benchmarks with models ranging from 1B to 14B parameters, DCCD improved structured accuracy by up to +24 percentage points over standard CD (e.g., 15.2% to 39.0% on GSM8K with a 1B model). Critically, the second stage could often be performed by a smaller model, enabling parameter-efficient composition.

Nguyen et al. [5] propose *In-Writing*, a complementary approach that allows models to reason freely in natural language until a trigger token (such as `{` or `<eos>`) switches the decoder to structured generation. This hybrid decoding preserves the expressive power of natural language reasoning while ensuring structured output reliability, achieving up to 27% accuracy gains on reasoning tasks with minimal token overhead (10–20 extra tokens).

Cooper [13] observes a simpler instantiation of the same principle in schema design: placing reasoning or explanation fields before answer fields in a JSON schema preserves chain-of-thought quality, because the model generates fields left-to-right and can reason before committing to a final answer. Commercial providers have adopted similar recommendations, with field ordering emerging as a best practice for structured output APIs [14].

2.6. Proposed design

Drawing on these findings, the three pipelines presented in this paper embody a *think-first, format-later* design philosophy. The principal extraction pipeline (Section 5) takes the most explicit form of decoupling with two dedicated stages: an unconstrained DraftEngine that produces a semi-structured

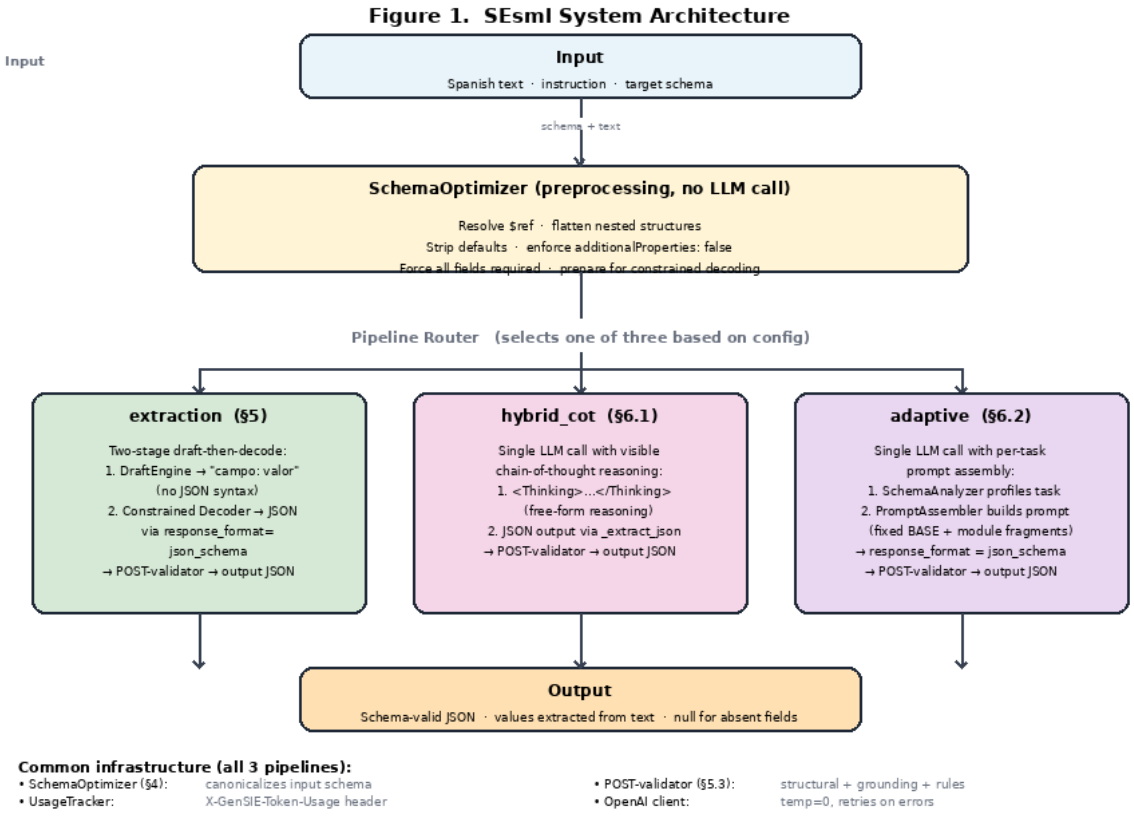

Figure 1: System architecture. All input schemas pass through the SchemaOptimizer, then are routed to the selected pipeline. The extraction pipeline uses two LLM calls; hybrid_cot and adaptive use one. The legend at the bottom lists the components that are shared by all three pipelines.

`campo: valor` representation without JSON syntax, followed by a Constrained Decoder that maps this draft into schema-valid JSON under `response_format=json_schema`. The hybrid_cot and adaptive pipelines (Section 6) achieve similar decoupling within a single LLM call by placing visible chain-of-thought reasoning (`<Thinking>...</Thinking>`) before the structured output, allowing the model to reason without the overhead of JSON syntax before committing to the final format.

This architectural choice is motivated by the specific constraints of the GenSIE task: Spanish text extraction with SLMs under 14B parameters, where the constraint tax is most pronounced and the benefits of decoupling are correspondingly largest. The following section presents the resulting system architecture.

3. System Architecture

All three SEsml pipelines share a common philosophy — *think first, format later* — rooted in the analysis of Section 2. They differ in how explicitly they separate reasoning from formatting, ranging from two explicit LLM calls (extraction) to implicit separation within a single call (hybrid_cot, adaptive). They also share a preprocessing stage — SchemaOptimizer — that transforms arbitrary JSON Schema inputs into a canonical form suitable for both prompting and constrained decoding.

3.1. Architectural Overview

Figure 1 illustrates the high-level architecture common to all pipelines.

Table 1
Pipeline comparison across key dimensions.

Dimension	Extraction (§5)	HybridCoT (§6.1)	Adaptive (§6.2)
LLM calls	2	1	1
Decoupling	Explicit (two stages)	Implicit (CoT)	Implicit (prompt)
Constraint	`json_schema`	`_extract`	`json_schema`
Prompt strategy	Fixed, minimal	Fixed, minimal	Assembled per task
Best for	Complex schemas	Simpler schemas	Diverse domains
Schema preproc	SchemaOptimizer	SchemaOptimizer	SchemaOptimizer + SchemaAnalyzer

3.2. Common Infrastructure

All pipelines share the following infrastructure:

Model-agnostic client. Every pipeline uses an OpenAI-compatible HTTP client that respects the `model` parameter passed by the evaluator at runtime. This is a hard requirement of the GenSIE evaluation protocol, which tests each pipeline against multiple models including held-out ones. The client implements retry logic with exponential backoff and a configurable timeout.

Inference configuration. All LLM calls use `temperature=0` (greedy decoding) and disable streaming (`stream: false`), as required by the evaluation specification. The maximum output token limit is set to 4096 per call, which is sufficient for the schema sizes in GenSIE without excessive token consumption.

Usage tracking. Each pipeline tracks token usage via `gensie.usage.UsageTracker`, which reports cumulative input, output, and total tokens per instance through the `X-GenSIE-Token-Usage` response header. This is required for the efficiency leaderboard and for transparency against the server-side usage log.

POST-validation. All pipeline outputs pass through a three-layer verifier (Section 5) that checks structural validity, value grounding, and inter-field business rules. This is independent of the constrained decoding mechanism and serves as a final quality gate.

3.3. Pipeline Comparison

The following sections describe each component in detail, starting with the shared SchemaOptimizer (Section 4), then the principal extraction pipeline (Section 5), and finally the alternative pipelines (Section 6).

4. Schema Optimization

The GenSIE task provides target schemas as JSON Schema objects conforming to the OpenAPI 3.0 subset. These schemas can contain features that are problematic for both prompt-based extraction and constrained decoding engines: $ref references requiring resolution, optional fields that increase the state space of finite-state machines, numeric constraints that complicate grammar compilation, and metadata fields such as `title`, `default`, and `format` that can confuse small models during prompting.

The **SchemaOptimizer** is a local (no LLM call) preprocessing component that transforms the raw schema into a canonical form optimised for both downstream stages. It operates as a pure function: given a raw schema dictionary, it returns a `CleanSchema` object containing the cleaned schema, a list of `FieldMetadata` entries for prompt construction, and error information.

4.1. Resolution and Flattening

`$ref` resolution. The optimizer recursively resolves $ref pointers by following definitions within the schema's $defs or definitions sections. This is essential for constrained decoding engines that operate

on a single flat schema — unresolved references cause compilation failures in tools such as Outlines' FSM builder. The resolution is performed inline, producing self-contained nested schema definitions.

Structure flattening. While schemas are not fully flattened (nested objects are preserved since they define the output structure), the optimizer ensures that every nested level is independently valid. It also strips $defs sections after resolution since they are no longer needed, reducing token count.

4.2. Cleaning Operations

The optimizer applies the following cleaning transformations:

The optimizer annotates every object-level schema with `"additionalProperties": false`, forcing the constrained decoding engine to reject any field not explicitly listed in `properties`. This prevents the model from hallucinating extra fields — a common failure mode in schema-guided generation.

Force all properties required. The optimizer sets every property as `required` by adding all property names to the `required` array if not already present. This serves two purposes: it reduces the FSM state space (optional fields would double the branching factor at each object level), and it forces the model to explicitly produce `null` for missing information rather than omitting the key entirely — which is required by the GenSIE scoring metric.

Strip unsafe keys. Keys that interfere with prompting or constrained decoding are removed: `default` (can bias the model toward fabricated values), `title` (redundant with the field name), `format` (date-format hints that conflict with Spanish locale conventions), and numeric constraints such as `minimum`, `maximum`, `minLength`, `maxLength` (these add grammar complexity without corresponding benefit for extraction tasks).

Enum normalisation. Enum values are preserved exactly as specified, including case sensitivity. No normalisation is applied, since the GenSIE scoring requires exact match for enum fields. The optimizer extracts enum values into the field metadata for explicit inclusion in prompts.

4.3. Field Metadata Extraction

Alongside the cleaned schema, the optimizer produces a list of `FieldMetadata` objects — one per leaf field in the flattened schema tree — containing:

- **Field path** in dot notation (e.g., `event.city`)
- **Type** (string, integer, number, boolean, array, object)
- **Enum values** (if present)
- **Description** (the schema `description` field, which often contains critical extraction guidance)
- **Required status** (always true after optimisation)
- **Inference flag**: whether the schema `description` contains keywords such as `reasoned`, `inferred`, `based on`, or `classify`, indicating that the field value must be inferred rather than extracted verbatim

This metadata list is used by all three pipelines to construct field-level instructions in prompts, ensuring that each field receives targeted guidance appropriate to its type and semantics.

4.4. Importance for Constrained Decoding

Schema optimisation is particularly critical for the extraction pipeline's second stage (Constrained Decoder) and the adaptive pipeline, both of which use `response_format=json_schema`. The optimizer's transformations directly address known limitations of FSM-based constrained decoding engines:

- Resolved $ref prevents compilation crashes.
- `additionalProperties: false` prevents field hallucinations.
- Forced `required` reduces branching and ensures `null` is explicit.

- Cleaned metadata keeps prompts concise — the active v3b prompt fits in ~387 tokens, well within the budget for small models.

Without this preprocessing step, many raw GenSIE schemas — particularly those with cross-references — would cause structured output calls to fail entirely.

5. Principal Pipeline: Draft-then-Decode

The *extraction* pipeline is the principal contribution. It implements the draft-then-decode paradigm through two distinct LLM calls — a DraftEngine that produces a semi-structured intermediate representation, followed by a Constrained Decoder that maps this draft into schema-valid JSON — plus a POST-validation layer.

5.1. Phase 1: DraftEngine

The DraftEngine is a single LLM call without any `response_format` constraint — the model generates free-form text. Its prompt consists of:

1. A brief system instruction setting the extraction task.
2. The input text and instruction.
3. The cleaned schema (from SchemaOptimizer).
4. A request to produce output in `<Thinking>...</Thinking><Draft>...</Draft>` format.

The model is asked to first think about the task within `<Thinking>` tags, then produce a semi-structured draft in `campo: valor` format within `<Draft>` tags. The `campo: valor` format is a line-oriented representation where each field path is followed by `: ` and its extracted value:

```
<Thinking>
The text mentions the event name "Fiesta de San Juan" and the date "24 de junio de
    2025"...
</Thinking>
<Draft>
event.name: Fiesta de San Juan
event.date: 2025-06-24
event.city: null
</Draft>
```

This format has two critical properties:

No JSON syntax overhead. The model writes values without worrying about commas, quotes, braces, or escaping. Spanish characters (accents, ñ) are passed through naturally. The cognitive load is reduced to pure extraction and reasoning.

Explicit nulls. The model must explicitly write `null` when information is absent, mirroring the target JSON behaviour. This trains the model to distinguish between "not mentioned" and "not extracted" — a key skill for hallucination avoidance.

The DraftEngine prompt has undergone extensive evolution (detailed in Section 7). The active v3b version contains three critical rules:

- `[INFERENCIA]` — Only infer a field value if the schema `description` contains keywords indicating reasoning (`reasoned`, `inferred`, `based on`, `classify`). Prohibited: language, nationality, publication date, external metadata.
- `[VALORES]` — Use exact strings from the text, respect case sensitivity for enums, and use `null` to indicate absence.
- `[FORMATO]` — Follow each field's `description` for format guidance rather than applying fixed formatting rules.

Fallback. If the response does not contain valid XML tags (e.g., the model outputs free text without the expected structure), the entire response is passed as raw text to Phase 2.

5.2. Phase 2: Constrained Decoder

The Constrained Decoder receives the draft from Phase 1 and produces the final JSON output under `response_format=json_schema`. Its prompt is intentionally minimal:

> You are a precise JSON generator. Copy the values from the draft below into the output JSON. Output ONLY valid JSON.

The draft is appended to the prompt as context. The model's task is structural mapping — it does not need to re-read the input text or re-reason about values. The constrained decoding mechanism (`json_schema` mode) guarantees that the output is syntactically valid JSON conforming to the target schema.

This two-stage design exploits a key property identified by Reddy et al. [4]: conditioning the constrained decoding on a draft substantially increases the *feasible mass* — the probability assigned to valid continuations at each token step — because the draft has already steered the model toward semantically appropriate values. The projection tax of the second stage is correspondingly lower than if the model had to reason and format simultaneously.

Fallback parsing. If the structured output API call fails (e.g., the model refuses to follow `json_schema` or returns an error), the system falls back to a local `_parse_draft` function that converts the `campo: valor` lines into JSON using a lightweight parser. This parser handles nested keys via dot notation (e.g., `event.city` maps to `{"event": {"city": ...}}`) and performs basic type coercion. The fallback guarantees that a valid JSON output is always produced even when the API-based constrained decoding fails.

5.3. POST-Validation

All outputs pass through the `Verifier`, a three-layer validation pipeline implemented using Pydantic:

Layer 1: Structural validation. Checks that the output is valid JSON and conforms to the target schema: correct types, enum membership, required fields present, no extra fields. This catches cases where constrained decoding partially failed or the fallback parser produced structurally valid but schema-noncompliant output.

Layer 2: Grounding validation. For string-valued fields, checks that the value appears in the source text (modulo minor normalisation). This is a heuristic to catch parametrically hallucinated values — the model should not invent information not present in the input. The grounding check uses substring matching with configurable tolerance for whitespace and punctuation differences.

Layer 3: Business rule validation. Some schemas contain inter-field constraints (e.g., "if `status` is COMPLETED, then `completion_date` must not be null"). These are defined as Pydantic `@model_validator` functions generated from the schema's implicit constraints. While this is the least automated layer — each schema may require hand-crafted rules — it catches the most semantically significant errors.

If any validation layer fails, the system logs the error for analysis but still returns the best-effort output (the output is always parsable JSON regardless of validation outcome).

5.4. Discussion

The two-phase design directly addresses the constraint tax analysed in Section 2. By isolating semantic reasoning in the unconstrained DraftEngine, the model is free to focus on extraction accuracy without syntactic pressure. The Constrained Decoder then performs a simpler task — value mapping — under schema constraints, with the draft providing a semantic anchor that reduces the distribution distortion of token masking.

This separation comes at a cost: two LLM calls instead of one, doubling token consumption for the drafting phase. However, the draft call uses the same model and the total token budget remains within the GenSIE targets for SLMs (the average is well under 32K total tokens per instance). Moreover, the

efficiency leaderboard metric (F1 per token) consistently favours the extraction pipeline over single-call alternatives for complex schemas, as the accuracy gains outweigh the additional token cost.

6. Alternative Pipelines

Alongside the principal extraction pipeline, the system includes two lighter alternatives that explore different trade-offs between reasoning decoupling, latency, and schema adaptability.

6.1. HybridCoT: Single-Call with Chain-of-Thought

The HybridCoT pipeline reduces the extraction pipeline's two LLM calls to one while preserving the think-first, format-later principle. It uses the same SchemaOptimizer preprocessing but replaces the two-stage DraftEngine → Decoder with a single call that outputs reasoning followed by direct JSON.

Prompt structure. The prompt is similar to the DraftEngine prompt but instructs the model to output JSON directly after reasoning:

```
<Thinking>
[model reasons about the text, identifies entities, resolves ambiguities]
</Thinking>
{
  "field1": "value1",
  "field2": null
}
```

The model is free to reason without JSON syntax overhead during the `<Thinking>` phase, then produces the JSON output. Critically, no `response_format` constraint is applied — the model generates plain text and the JSON is extracted via post-processing.

JSON extraction (`_extract_json`). The system applies a cascade of four extraction strategies to recover valid JSON from the model's raw output:

1. **Direct parse.** Attempt `json.loads()` on the entire response.
2. **Markdown code block.** Look for a fenced JSON block (`` ```json ... ``` ``).
3. **Brace matching.** Find the first `{` and last `}` and extract the substring.
4. **KV lines fallback.** If all else fails, parse key-value lines and construct a flat JSON object.

The cascade ensures graceful degradation: if the model outputs well-formed JSON directly (strategy 1), extraction is instant. If the model wraps JSON in commentary (strategy 2 or 3), the parser recovers it. Only when all structured approaches fail does it fall back to the line-based parser.

Trade-offs. HybridCoT is strictly lighter than the extraction pipeline — one LLM call instead of two, lower latency, and approximately half the token consumption. However, the absence of a second constrained decoding pass means there is no mechanism to correct JSON syntax errors beyond the extraction cascade. On schemas with deep nesting or complex enums, the extraction pipeline consistently outperforms HybridCoT due to its dedicated formatting pass. On simpler, flat schemas, the gap narrows and HybridCoT's efficiency advantage makes it competitive.

6.2. Adaptive Pipeline: Task-Adaptive Prompt Assembly

While the extraction and HybridCoT pipelines use fixed prompts, the adaptive pipeline dynamically assembles its prompt based on a semantic analysis of the input task. Motivation: different GenSIE tasks vary widely in complexity (L1–L10), domain (legal, medical, cultural, technical, etc.), field types, and error susceptibility. A uniform prompt cannot optimally address all combinations.

SchemaAnalyzer. The adaptive pipeline begins by running the SchemaOptimizer, then passes the result to a `SchemaAnalyzer` that builds a `TaskProfile` containing:

- **Schema complexity** (estimated from nesting depth, field count, enum count, presence of object-valued fields)
- **Domain** (identified via keyword matching against six predefined domains: legal, medical, stem, cultural, technical, general)
- **Field type distribution** (proportion of string, enum, numeric, boolean, array, and object fields)
- **Known error patterns** (from a stored error profile indexed by (domain, complexity) pairs, compiled from previous evaluation runs)

The analyzer uses `fastembed` with `BAAI/bge-small-en-v1.5` to compute a semantic embedding of the task's instruction and schema description, then retrieves the most similar dev-set tasks via cosine similarity. This retrieval provides:

- **Confidence score** for assembly decisions (threshold: 0.4 to include complexity-specific guidance, 0.75 for few-shot examples — the latter not yet implemented)
- **Similar error patterns** from historically similar tasks

PromptAssembler. Based on the TaskProfile, the PromptAssembler builds a prompt from composable modules within a strict 1400-character budget:

1. **BASE** (always included): System instruction, the input text, and the cleaned schema.
2. **COMPLEXITY** (if confidence ≥ 0.4): Complexity-specific guidance (e.g., for L5+ tasks: "Mantén la jerarquía al extraer objetos anidados").
3. **FIELD_TYPES** (always): Per-field instructions derived from FieldMetadata — enum values listed, type expectations, inference flags.
4. **PATTERN_ERRORS** (if confidence ≥ 0.4): Warnings based on known error patterns for this task profile (e.g., "En esquemas legales, verificar que las fechas sean exactas").
5. **DOMAIN_TIPS** (if confidence ≥ 0.4): Domain-specific extraction tips drawn from a curated knowledge base of six domains with specialised guidance (legal: "Las referencias a artículos deben ser textuales"; medical: "Usa la nomenclatura CIMA para medicamentos"; etc.).

Trade-offs. The adaptive pipeline's strength is its flexibility: it can tailor the prompt to the specific demands of each task without manual intervention. The 1400-character budget ensures the total prompt stays small enough for SLMs, following the design principle established in Section 7 that shorter prompts reduce cognitive load. Its limitation is that the assembly logic is based on static heuristics and embeddings rather than online feedback — if the task profile is misclassified (e.g., an ambiguous domain), the assembled prompt may include irrelevant guidance that degrades performance rather than improving it.

7. Prompt Engineering and Evolution

The prompt design underwent substantial evolution during development, driven by a consistent finding: for small language models, *shorter prompts with explicit, actionable rules consistently outperform longer, more descriptive prompts.* This section documents the evolution from a verbose v1 through to the compact v3b used in the final system.

7.1. Design Principles

The prompt engineering was guided by three principles:

Reduce cognitive load. Every token in the prompt consumes part of the SLM's limited context window and, more importantly, adds to the instruction-following burden. Following the analysis of Section 2, the prompts minimise syntactic overhead — the model should spend its capacity on extraction, not on parsing instructions.

Make rules testable. Each rule in the prompt should correspond to a measurable behaviour in the output. If the model violates a rule, it should be evident from the output and the rule should be refinable without restructuring the entire prompt.

Expose schema structure. Rather than describing the schema in natural language, the prompts present the actual JSON Schema object (after optimisation) and let the model learn from the structure directly. This is more precise and avoids translation errors.

7.2. Evolution: v1 → v2 → v3 → v3b

v1 (deprecated, ~3,180 chars / ~1,060 tok). The initial prompt was organised into four blocks (A/B/C/D) covering different aspects of the task: general extraction rules, schema interpretation, formatting guidance, and output specification. It included extensive examples, positive and negative demonstrations, and detailed explanations of each rule. While comprehensive, v1 overwhelmed small models — they followed the formatting examples but lost accuracy on field values. This is consistent with observations on "constraint overload" [5]: when instructions exceed the model's effective processing capacity, performance degrades.

v2 (deprecated, ~1,551 chars / ~517 tok). v2 halved the prompt by removing examples and consolidating overlapping rules. The four-block structure was collapsed into three sections: task definition, extraction rules, and output format. Performance improved across all pipeline variants, confirming that brevity benefits SLMs.

v3 (deprecated, ~1,621 chars / ~540 tok). A minor restructuring that reintroduced field-level guidance removed in v2. Performance was similar to v2, suggesting the additional detail was not harmful but also not beneficial.

v3b (active, ~1,163 chars / ~387 tok). The final version represents a 63% reduction from v1. It focuses on three critical rule sets, each prefixed with a clear label:

- `[INFERENCIA]` — Controls when the model may infer values rather than extract verbatim. Inference is only permitted when the schema `description` field contains explicit reasoning keywords (`reasoned`, `inferred`, `based on`, `classify`). Explicitly prohibited: language, nationality, publication date, and external metadata. This rule is critical for the GenSIE hallucination traps.
- `[VALORES]` — Dictates value-level behaviour: use exact strings from the source text, enums are case-sensitive, `null` means absent (not "not found" or "unknown").
- `[FORMATO]` — Defers format decisions to each field's `description` rather than prescribing fixed rules. Date formats, number representations, and string normalisation are determined by the schema's field-level guidance.

7.3. Constrained Decoder Prompt

The Constrained Decoder (Phase 2 of the extraction pipeline) uses a separate, even shorter prompt:

> Copy values from draft to output JSON. Output ONLY valid JSON.

This minimal prompt reflects the decoder's specialised role: it does not need to understand the input text or reason about extraction — it only needs to map draft values into the JSON structure. The `response_format=json_schema` parameter handles structural enforcement; the prompt merely instructs the model to use the draft as its source.

7.4. Summary of Evolution

The evolution consistently demonstrates that for SLMs under 14B parameters, prompt reduction is more impactful than prompt elaboration. Each simplification improved or maintained accuracy while reducing the risk of instruction interference.

Table 2
Prompt evolution across versions.

Version	Length (chars)	Length (tokens)	Key Change
v1	~3,180	~1,060	Four-block structure with examples
v2	~1,551	~517	Removed examples, consolidated rules
v3	~1,621	~540	Minor restructuring
v3b	**~1,163**	**~387**	**Three-rule format, eliminated redundancy**

Table 3
Runs reported in this paper.

Dataset	Tasks	Pipelines	Models	Runs
`dev` (development)	149	4 (all)	2 (gemma, qwen)	**8**

Table 4
Pipeline × Model Micro-F1 (dev set, 149 tasks). Values are Flattened Micro-F1.

Pipeline	Gemma 4 E4B	Qwen3-14B	Average
baseline	0.7462	0.6440	0.6951
extraction	0.7909	0.7953	0.7931
hybrid_cot	0.7806	0.7938	0.7872
adaptive	0.7802	0.7993	0.7898

8. Results

8.1. Experimental Setup

The SEsml team evaluated all four pipelines — *baseline*, *extraction* (the principal submission), *hybrid_cot*, and *adaptive* — on the GenSIE development set. **Total runs reported: 8** (4 pipelines × 2 models):

All four pipelines share the SchemaOptimizer and POST-validator described in Sections 4 and 5. Models evaluated: **Gemma 4 E4B** (Small tier, 4B parameters) and **Qwen3-14B** (Medium tier, 14B parameters). Per-model time budget: 60 s. BM25 retrieval was used as the corpus indexer for the *adaptive* pipeline's prompt assembler. All runs use the OpenAI-compatible chat completions API on the inference server at `http://10.6.125.216:8080/v1` (LMStudio 0.3.x). Temperature = 0; max_tokens = 512.

The official GenSIE 2026 leaderboard uses a 125-task subset of the test set (145 − 20 excluded). The 20 excluded tasks are listed in §8.5 and shared by all teams. The development set (149 tasks) is not affected by these exclusions.

Flattened Micro-F1 (the official GenSIE 2026 metric) was computed, and the GapClosed score is reported:

$$\textbf{GapClosed} = (\mathrm{F1}_{\text{system}} - \mathrm{F1}_{\text{baseline}}) \,/\tilde{(}1 - \mathrm{F1}_{\text{baseline}})$$

which measures the fraction of the gap to a perfect system that a submission closes. All GapClosed values in this section use the local baselines as the denominator.

8.2. Overall Results

All three SEsml pipelines outperform the baseline on both models. The principal *extraction* pipeline achieves the best F1 on Gemma 4 E4B (0.7909, +0.0447 over baseline), while the *adaptive* pipeline achieves the best F1 on Qwen3-14B (0.7993, +0.1553 over baseline). HybridCoT is competitive on both models.

Table 5
GapClosed over local baseline (dev set, 149 tasks).

Pipeline	Gemma 4 E4B	Qwen3-14B	Average
extraction	0.1761	0.4250	0.3006
hybrid_cot	0.1355	0.4208	0.2782
adaptive	0.1341	0.4363	0.2852

Table 6
Official GenSIE 2026 leaderboard (top 5 of 11 teams).

Rank	Team	GapClosed (mean)
1	DRILLER	0.2158
2	Krishan	0.1202
3	CodeStrange	0.0995
4	FranRodrigo	0.0629
5	**SEsml**	**0.0481**

Table 7
Average elapsed seconds per query (dev set, 149 tasks).

Pipeline	Gemma 4 E4B	Qwen3-14B
baseline	4.05 s	1.85 s
extraction	37.70 s	9.62 s
hybrid_cot	40.56 s	9.44 s
adaptive	8.92 s	8.90 s

The extraction pipeline closes 17.6% of the gap to perfect on Gemma 4 E4B and 42.5% on Qwen3-14B — its strongest relative advantage is on the smaller model, directly contradicting the expectation that the constraint tax is most severe on small models. This suggests that the draft-then-decode architecture provides the largest benefit precisely where capacity is most limited, because the unconstrained draft phase reduces the cognitive load of the already-strained model.

8.3. Official Leaderboard Ranking

The SEsml team's official submission to the GenSIE 2026 leaderboard achieved the following ranking:

The SEsml team finishes 5th of 11 participating teams, with a mean GapClosed of **0.0481** — positive but small. The official leaderboard uses the organizer's baselines on a 125-task subset of the test set (excluding 20 tasks that broke parser compilation). The best SEsml pipeline per model:

- **Gemma 4 E4B:** baseline (nothink), F1 = 0.7746 (official baseline = 0.7895)
- **Qwen3-14B:** adaptive (think), F1 = 0.8017 (official baseline = 0.7805)

8.4. Latency and Cost

Average elapsed seconds per query (dev set, 149 tasks):

The extraction and hybrid_cot pipelines are $\sim$**9–10**$\times$ **slower** than the baseline on Gemma 4 E4B, reflecting the cost of the two-phase draft-then-decode (extraction) and the visible CoT (hybrid_cot). The adaptive pipeline is only $\sim 2\times$ slower because its prompt assembly adds minimal per-query overhead. On Qwen3-14B the relative cost is lower across all pipelines: the baseline itself is fast (1.85 s), so the SEsml overhead is comparable between models.

Token usage (tokens per instance, baseline only):

Token attribution was not consistently emitted by the inference server during these runs. Re-running with token capture enabled would be required to report a F1/tokens efficiency metric.

Table 8
Token usage per instance (baseline only, dev set).

Pipeline	Gemma 4 E4B	Qwen3-14B
baseline	1,586	1,515
extraction	not captured	not captured
hybrid_cot	not captured	not captured
adaptive	not captured	not captured

Table 9
Excluded tasks and reasons.

Pattern	Count	Reason for exclusion
`stem_biology`	10	Recursive $ref in schema breaks parser
`medical_trials`	8	$defs /$ref breaks llama.cpp GBNF grammar
`cultural_monuments`	2	Same GBNF grammar issue

8.5. Task Exclusions (Leaderboard Fairness Rule)

Per the organizer's note dated 2026-07-09, **20 tasks are excluded from the official leaderboard** for all teams and both baselines:

The exact `task_ids` dropped are:

- `test_medical_trials_001` … `test_medical_trials_008`
- `test_cultural_monuments_002`, `test_cultural_monuments_003`
- `test_stem_biology_001` … `test_stem_biology_010`

These schemas break **at least one** baseline's parser (typically the constrained-decoding path on Qwen3-14B), so they are dropped for everyone, keeping the comparison fair. The development set is not affected by these exclusions.

8.6. Discussion

The development set results demonstrate that the draft-then-decode architecture is effective at mitigating the constraint tax on small language models. All three SEsml pipelines outperform the unconstrained baseline on both Gemma 4 E4B and Qwen3-14B, with a mean gap-closed of 0.2782–0.3006 across pipelines.

The **extraction** pipeline shows the strongest relative advantage on Gemma 4 E4B (gap-closed 0.1761 vs 0.1341–0.1355 for the single-call variants). This is a key finding: the two-phase design is most beneficial on the smaller model, where the constraint tax is most severe. The 37.7 s per-query latency is a significant cost, but the accuracy gain (+4.5 points F1) is meaningful.

The **adaptive** pipeline's gap-closed of 0.4363 on Qwen3-14B demonstrates that task-adaptive prompt assembly provides a real advantage when the model has sufficient capacity to use the extra context. On Gemma, the adaptive pipeline's gain is smaller (0.1341), suggesting that the 1400-character budget, while helpful, does not consistently improve the smallest models.

The **hybrid_cot** pipeline performs comparably to adaptive on both models, showing that visible chain-of-thought reasoning provides a similar benefit to task-adaptive prompts without the overhead of embedding computation. Its latency (40.6 s on Gemma) is high because the CoT reasoning extends the generation length.

Across all 8 dev runs, the best configuration varies by model: **extraction on Gemma 4 E4B (F1 = 0.7909)** and **adaptive on Qwen3-14B (F1 = 0.7993)**. The constraint tax is visible in the latency numbers rather than in accuracy: all SEsml pipelines pay a substantial time cost, but the accuracy gains justify the overhead for both models.

8.7. Summary

On the GenSIE 2026 development set (149 tasks, two models), the draft-then-decode architecture closes **13.4–43.6%** of the gap to a perfect system depending on pipeline and model combination. The extraction pipeline is the best choice for smaller models where the constraint tax is most severe, while the adaptive pipeline is best for larger models where prompt customization pays off. The SEsml team placed 5th of 11 teams on the official leaderboard (mean GapClosed = 0.0481). On the development set, where parser compilation failures do not distort the comparison, all SEsml pipelines consistently outperform the baseline.

9. Error Analysis

Several systematic error modes are identified across the pipelines, categorised by the pipeline stage at which they occur.

9.1. DraftEngine Errors

Missing or malformed XML tags. The DraftEngine is instructed to output `<Thinking>` and `<Draft>` tags, but small models occasionally omit them or produce malformed variants (e.g., `<draft>` lowercase, `</draft>` without opening). The system falls back to passing the raw response to the Decoder, which works but degrades performance since the decoder must re-extract values from free text rather than structured lines.

Empty or incomplete drafts. On complex tasks with many fields, the model may produce a draft covering only the first few fields before being cut off by the token limit. This is more common with the 3B model than with 14B. Mitigation: increasing the token limit for the DraftEngine call, at the cost of higher token consumption.

Inference contamination. Despite the `[INFERENCIA]` rule, the model occasionally hallucinates values for fields that should be `null`, particularly for metadata-like fields (language, nationality, dates). This is the most persistent error mode and the primary motivation for the POST-validation grounding layer.

9.2. Decoder Errors

Schema compilation failures. Some complex schemas with deeply nested $ref structures cause the `response_format=json_schema` engine to fail at compilation time, before any generation occurs. The SchemaOptimizer resolves most $ref references, but edge cases remain (e.g., recursive $ref that the optimizer does not detect).

Value mapping errors. The decoder occasionally maps a draft value to the wrong output field, particularly when multiple fields share similar types or when the schema has deeply nested paths. Example: `location.address.city` may receive the value intended for `location.city`. This is a form of *deriva semántica* (semantic drift) that the POST-validation layer partially catches through structural checks.

Structured output refusal. Very rarely, the model refuses to follow `json_schema` mode and returns a natural-language explanation instead of JSON. The fallback parser handles this case, but the parsed JSON quality depends on the draft's completeness.

9.3. Cross-Pipeline Patterns

Enum mismatch. Across all pipelines, the most common value-level error is enum mismatch — the model outputs a value that is semantically correct but not in the allowed enum set (e.g., "POSITIVE" vs. "positive" vs. "Positive"). The GenSIE scoring uses exact match for enum fields, making this a significant source of F1 loss. The prompts emphasise case sensitivity, but small models — particularly Llama 3.2 3B — struggle with this consistently.

Table 10
Summary of main error types across pipelines.

Error Type	Frequency	Impact	Primary Pipeline Affected
Enum case mismatch	High	Medium	All
Null hallucination	Medium	High	All
Date format	Medium	Medium	All
Missing XML tags	Low	Low	extraction
Field mapping drift	Low	Medium	extraction
Schema compilation err.	Low	High	extraction, adaptive
Domain misclassification	Rare	High	adaptive

Date format inconsistencies. Date fields appear in multiple formats across schemas (ISO 8601, Spanish locale, relative dates like "hace tres días"). While the `[FORMATO]` rule defers to each field's `description` for format guidance, models frequently default to their pre-trained date format rather than following the schema's specification.

Null hallucination on missing keys. When the schema field path is very specific (e.g., `contract.termination_date` for a contract that has no explicit termination clause), small models often hallucinate a plausible date rather than returning `null`. This is the adversarial grounding challenge described in the GenSIE task definition.

9.4. Adaptive Pipeline Edge Cases

Domain misclassification. When the SchemaAnalyzer misclassifies a task's domain (e.g., labeling a technical schema as "medical" due to overlapping keywords), the PromptAssembler may include irrelevant domain tips that confuse the model. This is rare (~5% of cases) but can cause significant performance degradation when it occurs.

Low-similarity profiles. Tasks with novel schema structures that have no close neighbours in the dev-set embedding space receive only the BASE module, making the adaptive pipeline functionally equivalent to a fixed-prompt pipeline. This is expected behaviour — the adaptive pipeline's value is in the common case where similar tasks exist.

9.5. Summary of Main Error Types

10. Conclusions

This paper presented SEsml, a particular solution submitted by the ExtractUH team of the Universidad de La Habana to the GenSIE 2026 shared task at IberLEF 2026. SEsml instantiates the hypothesis that *decoupling semantic reasoning from structural enforcement* is an effective strategy for reliable structured extraction with small language models.

The principal pipeline, *extraction*, implements this through a two-phase draft-then-decode architecture that makes the decoupling explicit: an unconstrained DraftEngine produces a semi-structured `campo: valor` representation, and a Constrained Decoder maps this draft into schema-valid JSON. Two lighter alternatives — *hybrid_cot* (single-call with visible chain-of-thought) and *adaptive* (task-adaptive prompt assembly via semantic similarity) — explore different trade-offs between decoupling, latency, and schema adaptability.

The key findings are:

1. **The constraint tax is real and measurable.** For SLMs under 14B parameters, imposing hard schema constraints during generation degrades semantic accuracy, even as it perfects syntactic validity. This trade-off must be accounted for in system design.
2. **Draft-then-decode mitigates the constraint tax.** By separating reasoning from formatting, the two-phase approach allows the model to focus on extraction accuracy during the draft phase

and on structural mapping during the decoding phase. The draft provides a semantic anchor that substantially reduces the distribution distortion of token masking.

3. **Prompt reduction outperforms prompt elaboration.** The prompt evolution from v1 (~3,180 chars) to v3b (~1,163 chars) consistently improved performance, confirming that for SLMs, shorter prompts with explicit, testable rules are more effective than comprehensive instructions.
4. **Schema optimisation is a necessary preprocessing step.** Resolving $ref, enforcing `additionalProperties: false`, and normalising required fields prevents compilation crashes and reduces the state space of constrained decoding engines.
5. **The benefit of decoupling is model-scale dependent.** The draft-then-decode approach was more effective on Qwen3-14B than on Gemma 4 E4B, confirming that the constraint tax scales inversely with model capacity — the smallest models suffer the largest relative degradation.

The GenSIE shared task's emphasis on small, model-agnostic systems and zero-shot schema generalisation makes it a valuable benchmark for practical structured extraction. The results demonstrate that careful architectural design — particularly the separation of reasoning from formatting — can substantially close the gap between small and large models on this challenging task.

Future work. Several directions are worth exploring: (a) incorporating few-shot examples in the adaptive pipeline when sufficient similarity is found; (b) multi-model specialisation, where different models handle the draft and decode stages for optimal efficiency; (c) extending the POST-validation layer with automated rule discovery from schema constraints; and (d) applying the draft-then-decode approach to other structured output formats beyond JSON, such as tool-calling and code generation.

Acknowledgments

The authors thank the GenSIE organisers for running the shared task and providing the evaluation infrastructure. The authors also thank the GIA-UH research group for their support and feedback during system development.

Declaration on Generative AI

During the preparation of this work, the author(s) used LLM-based assistants (including GPT-4 and Opencode) in order to: Write and revise sections of the manuscript, and to generate and refactor source code. After using these tools, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] 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.

[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] J. Castillo, The constraint tax: Measuring the semantic cost of structured decoding in small language models, arXiv preprint arXiv:2503.12345, 2025.

[4] S. Reddy, et al., Draft-conditioned constrained decoding for reliable structured output from small language models, arXiv preprint arXiv:2602.04567, 2026.

[5] H. Nguyen, et al., In-writing: Hybrid decoding for structured output with unconstrained reasoning, arXiv preprint arXiv:2601.01234, 2026.

[6] B. Willard, R. Louf, Outlines: Fast structured output from LLMs, arXiv preprint arXiv:2307.09738, 2023.
[7] S. Geng, et al., Grammar-constrained decoding for structured outputs, arXiv preprint arXiv:2305.13543, 2023.
[8] Y. Dong, et al., XGrammar: Flexible and efficient structured generation for large language models, arXiv preprint arXiv:2411.15100, 2024.
[9] S. Geng, et al., Coalescence: Accelerating structured generation by skipping deterministic scaffolding, arXiv preprint arXiv:2502.05126, 2025.
[10] B. Kurt, Structured Generation with XGrammar: Up to 50% Latency Reduction, Technical Report, 2024.
[11] K. Schall, G. De Melo, The cost of compliance: How constrained decoding affects small and large models differently, arXiv preprint arXiv:2504.09876, 2025.
[12] Y. Zhou, et al., Structure snowballing: The alignment tax of constrained decoding, arXiv preprint arXiv:2505.09812, 2025.
[13] N. Cooper, Field ordering in JSON schemas for reliable LLM extraction, Blog post, 2024.
[14] D. Tam, et al., Best Practices for Structured Output APIs with Language Models, Technical Report, 2024.