CodeStrange at GenSIE 2026: Metric-Aware Hybrid Pipelines for Spanish Schema-Guided Information Extraction

Leynier Gutierrez González[1], Carlos Bermudez Porto[2] and Roberto Marti Cedeño[3,*]

[1]Independent researcher, Mexico City, Mexico
[2]Independent researcher, Havana, Cuba
[3]University of Havana, Havana, Cuba

Abstract

GenSIE asks systems to extract grounded JSON objects from Spanish texts when the target schema is supplied only at inference time. This setting combines zero-shot schema understanding, nested structured generation, hallucination control, and efficiency under small and medium language models. This paper describes the CodeStrange submission, a portfolio of three inference-time pipelines: a guard-first ReAct repair pipeline, a grounding-focused pipeline with evidence and recovery passes, and a single-call spaCy evidence pipeline. We align the system with the state of the art in generative information extraction, constrained structured output, validation and repair, retrieval-augmented extraction, and hybrid NLP. We also report the experimental trajectory that led to the final system, including baseline studies, historical ablations, grounded-adaptive variants, hard-case re-benchmarking, and the final no-think self-evaluation on `data/test` with Gemma 4 E4B and Qwen3 14B. Guard-first repair was the strongest submitted pipeline on both models, reaching 0.8091 Micro-F1 on Gemma and 0.8003 on Qwen, with an average gap-closed score of 35.2%. spaCy evidence reached a 27.1% average gap-closed score, while `grounded` was mixed: below baseline on Gemma but above baseline on Qwen. The results show that the most reliable gains came from conservative schema control: schema verbalization, JSON-only prompting, deterministic cleanup, explicit null discipline, and validation-triggered repair. We conclude with limitations caused by contest metric changes and provide recommendations for future schema-guided extraction systems.

Keywords

Information extraction, Schema-guided extraction, Structured output generation, JSON Schema, Spanish NLP, Small language models, Grounded generation

1. Introduction

The GenSIE shared task evaluates general-purpose schema-guided information extraction from Spanish text as part of IberLEF 2026 [1, 2]. For each instance, a system receives a source document, a natural-language instruction, and a target schema represented as a JSON Schema/OpenAPI subset. The system must return a valid JSON object whose values are grounded in the text and whose structure follows the schema [3]. Unlike conventional information extraction benchmarks, the schema is not fixed: systems must infer field meaning, type constraints, nullability, enum semantics, and nesting patterns at runtime.

The task is difficult for small and medium open-weight models for four reasons. First, models must interpret dynamic schemas instead of relying on a fixed ontology. Second, they must generate machine-readable JSON without malformed syntax, extra prose, or structural drift. Third, they must distinguish extractive fields from inferential fields such as legal outcomes, trial success labels, media verdicts, or recipe complexity. Fourth, they must return null when a value is not supported by the supplied context, even if the answer is widely known outside the text. This last requirement turns GenSIE into a grounding benchmark as much as an information extraction benchmark.

CodeStrange approached GenSIE as an inference-time engineering problem. We did not fine-tune model weights. Instead, we explored prompt compilation, deterministic decoding, structured output

IberLEF 2026, September 2026, León, Spain
*Corresponding author.
rmarticedeno@matcom.uh.cu (R. M. Cedeño)
0000-0001-6393-1679 (L. G. González); 0000-0002-5267-0327 (C. B. Porto); 0000-0002-7671-4187 (R. M. Cedeño)

modes, JSON repair, schema-aware validators, evidence extraction, lightweight retrieval, and budget-aware repair loops. The final contest entry exposes three pipelines through the official participant interface:

1. `combo_guard_react_repair_ref`, a guard-first pipeline with deterministic validation and repair only when quality checks require it;
2. `grounded`, a grounding-focused pipeline with schema digests, optional evidence gathering, long-text passage retrieval, robust JSON repair, and targeted field recovery;
3. `combo_guard_list_spacy_ref`, a single-call pipeline that augments the prompt with schema-routed spaCy and regular-expression evidence.

This paper contributes a technical description of the submission, a state-of-the-art alignment between GenSIE and the implemented techniques, and a report of the experimental evidence preserved in the repository. We deliberately separate experimental phases because the contest protocol and metrics changed during development. In particular, the final metric version was not tested as thoroughly as earlier versions because of the gap between publication timing and the camera-ready deadline. The repository snapshot now includes final Gemma 4 E4B and Qwen3 14B no-think self-evaluation reports for `data/test`. We therefore report final-metric-compatible numbers where available, while treating them as local saved evidence rather than official leaderboard scores.

2. Related Work and Methodological Positioning

2.1. Generative Information Extraction

Traditional information extraction systems were usually specialized to stable schemas: named entity recognition, relation extraction, event extraction, slot filling, or domain-specific record construction. Rule-based systems offered transparency and control but were brittle under domain shift. Neural systems improved robustness, yet generally required labeled examples for each target schema or domain. Recent work reframes extraction as conditional generation with pretrained language models and large language models [4, 5]. The model receives an instruction and emits a structured representation. This paradigm matches GenSIE's dynamic-schema setting but introduces instability in formatting, grounding, completeness, and schema adherence [6].

GenSIE sits at the intersection of generative information extraction, structured output generation, and zero-shot schema adaptation. It is also multilingual and efficiency-aware: the data are Spanish, and evaluation emphasizes small and medium models rather than frontier proprietary systems. For CodeStrange, this made pure prompt engineering insufficient. The practical system had to surround the model with schema-aware scaffolding, validators, repair mechanisms, and evidence control.

2.2. Structured Output, Validation, and Repair

Structured output generation is now central to language-model agents, tool use, and schema-constrained extraction. JSON Schema is a natural representation because it supports field names, nested objects, arrays, required fields, nullable fields, enums, and scalar types. Recent structured-output studies, including JSONSchemaBench, show that constrained decoding can improve syntactic validity while leaving semantic correctness and efficiency unresolved [7]. Grammar engines such as XGrammar show that formal constraints can be integrated into LLM serving with low overhead, but they still mainly guarantee shape rather than factual extraction quality [8]. SLOT similarly frames structured output as a core capability for agentic systems and extraction workflows [9].

The CodeStrange system therefore treats structured output as a layered problem. Some pipelines use OpenAI-compatible `json_schema` or `json_object` response formats when supported by the backend. Others use text output plus `json_repair` [10]. All final pipelines apply deterministic postprocessing: pruning unknown keys, normalizing enums, coercing scalar types, preserving required keys, deduplicating lists, and handling optional null fields according to schema conventions. This

Table 1
Mapping from state-of-the-art techniques to CodeStrange implementation mechanisms.

Technique family	CodeStrange implementation	Purpose in GenSIE
Schema-guided structured output	`json_schema`, `json_object`, raw text JSON, `json_repair`, dynamic Pydantic-style validation	Reduce malformed output and recover partial credit from imperfect model responses.
Schema verbalization	Recursive schema digests, field checklists, enum/null rule blocks	Help small models interpret dynamic schemas supplied only at inference time.
Grounded prompting	Spanish system prompts, explicit null rules, evidence-first prompts, no-external-knowledge instructions	Reduce hallucinated non-null values and unsupported completions.
Hybrid NLP evidence	spaCy entities, noun chunks, sentence evidence, regex dates/numbers/law references	Provide cheap field-level candidate spans and improve grounding.
Retrieval augmentation	`fastembed` passage selection for long-text tails in `grounded`	Recover relevant content after truncation while controlling context length.
Self-correction / ReAct	Deterministic issue detection, repair prompts, critic/field-QA ablations, budget gates	Repair missing required fields, enum errors, unsupported values, and low-recall lists only when needed.
Efficiency control	Temperature 0 defaults, call caps, token tracking, time budgets, targeted recovery	Preserve performance while respecting GenSIE token and timing constraints.

combination reflects a core lesson of the experiments: constrained decoding is valuable, but it is not a substitute for semantic validation and repair.

2.3. Grounding, Retrieval, and Hybrid NLP

Grounded extraction is central to GenSIE. The system must avoid filling missing fields with parametric knowledge, and it must avoid over-generating unsupported values. Retrieval-augmented and example-augmented extraction methods are therefore relevant, provided that they are transparent and do not overfit to development schemas [11]. Schema-guided agentic extraction systems such as OneKE similarly emphasize configuration, debugging, and correction around a schema rather than a single unguarded generation step [12]. Classical NLP remains useful as a low-cost evidence layer: spaCy provides sentence segmentation, named entities, noun chunks, and rule-based matching components that can surface candidate spans before or during model generation [13, 14].

The repository contains both lightweight and heavier grounding mechanisms. The `grounded` pipeline uses schema digests, evidence notes, optional embedding-based passage retrieval for truncated long texts, and targeted recovery passes. The spaCy pipeline builds a compact evidence pack per top-level field using roles such as named entity, numeric literal, date literal, array extraction, enum/boolean, and free-text evidence. The ReAct family performs validation-driven repair and, in ablations, field-level QA, schema optimization ideas, and consensus-style checks [15, 16]. Table 1 summarizes how SOTA ideas map to concrete CodeStrange implementation choices.

3. System Architecture

3.1. Task Interface and Instrumentation

All pipelines implement the `GenSIEAgent` interface and are served through the `OfficialParticipant`. A task contains an identifier, an instruction, an input text, and a target schema. The participant exposes pipeline metadata through `/info` and routes `/run` requests to the selected agent. The implementation records per-instance wall time, model-call time, non-LLM

reasoning time, call count, and token usage when the backend supplies OpenAI-compatible usage metadata. The evaluator stores these values in JSON reports, making it possible to compare not only F1 but also latency and token cost.

The shared evaluation logic uses flattened schema scoring. Gold and system JSON objects are recursively compared by field path. Rigid fields such as numbers, booleans, date-like strings, and enums require exact matches. Free text uses a hybrid lexical/semantic similarity. Lists are compared with order-independent greedy matching. Corpus-level precision, recall, and Micro-F1 are computed from total true-positive score, total gold key count, and total system key count. Later contest updates added gap-closed ranking over baseline and softened token/time budgets, which affected the interpretation of earlier runs.

3.2. Common Postprocessing

The pipelines share several postprocessing principles. Generated JSON is parsed or repaired; unexpected keys are pruned when the schema disallows additional properties; scalar values are coerced when unambiguous; enum strings are normalized to exact schema values; dates and Spanish numeric formats are normalized in selected agents; duplicate list items are removed; and optional null handling is adjusted to avoid penalizing systems for emitting unsupported optional fields. These steps are intentionally conservative: they repair syntax and schema shape but do not invent facts.

3.3. Pipeline 1: Guard-First ReAct Repair

`combo_guard_react_repair_ref` is a budget-aware validation and repair pipeline. The first pass is a guarded draft at temperature 0. The prompt emphasizes exact enums, nulls for unsupported values, list recall, and JSON-only output. The raw response is parsed with `json_repair` and sanitized to the schema. A deterministic validator then detects issue kinds such as missing required fields, extra keys, enum invalidity, type mismatch, empty required arrays, unsupported non-null values, suspicious nulls, and low-recall lists.

The key design choice is selective escalation. If no high-value issue kinds remain, the first draft is returned. If escalation is justified and the call/time budget permits, the agent asks the model to repair only the marked paths while preserving the rest of the object. This avoids the cost of unconditional multi-step reasoning. The maximum call count for the final submitted guard-repair variant is three, and token budgeting begins after the first call. The pipeline therefore operationalizes ReAct-style verification and correction without allowing open-ended agent loops.

3.4. Pipeline 2: Grounded Extraction

`grounded` is the general grounding-focused pipeline. It converts the schema into a recursive field digest indicating field paths, requirement status, nullability, enum/type information, and short descriptions. The extraction prompt places the instruction, schema, and text in a fixed order and includes a checklist for required and must-output fields. For complex schemas, the agent can run an evidence pass that asks the model to cite relevant text fragments before extraction. For long texts, it truncates the head of the document and optionally uses `fastembed` to retrieve high-relevance chunks from the omitted tail. Heavy-output schemas may skip the evidence pass to avoid timeouts.

The primary extraction uses `json_object` when possible and falls back to text. If parsing fails, the pipeline performs a repair call. After parsing, it corrects verbatim spans where possible, cleans strings, coerces scalar types, strips optional nulls under schema-aware rules, prunes extra keys, and normalizes enums. A targeted recovery pass can fill empty include-if-found fields when the time budget permits. This makes `grounded` the broadest pipeline: it combines schema verbalization, grounding cues, retrieval for long texts, robust decoding, and field-level recovery.

3.5. Pipeline 3: Guard/List spaCy Evidence

`combo_guard_list_spacy_ref` is a single-call evidence pipeline. It parses the schema, routes top-level fields into evidence roles, and builds a bounded evidence pack from spaCy entities, noun chunks, sentences, and regular expressions for dates, percentages, numbers, temperature, money, legal references, and articles. The prompt includes enum/null rules, list-recall rules, per-field evidence, grounding rules, and a JSON-only output requirement. The model is called once at temperature 0 with text output, parsed by `json_repair`, and postprocessed recursively.

This pipeline is attractive for efficiency because it uses one model call and pushes evidence construction outside the LLM. However, experiments revealed a model-specific failure mode: on Qwen 1.7B, the model sometimes copied or transformed the evidence-pack schema itself into the answer, returning nested objects such as `{"role": ..., "candidates": ...}` as field values rather than extracting the target JSON. On Gemma 4B, the same evidence pack improved hard-case performance. This makes the pipeline a useful example of both the promise and fragility of hybrid evidence prompting.

4. Experimental Methodology

4.1. Datasets and Models

The repository contains several experimental phases. The starter set has 40 instances, the development set has 149 saved instances in the evaluated reports, and a curated hard-case set has 74 instances selected from starter and development tasks that were difficult for multiple baselines. Additional historical hard subsets and regression gates are preserved in `results_old/experiments`. The evaluated models include `qwen/qwen3-1.7b`, `google/gemma-4-e2b`, `llama-3.2-3b-instruct`, and, in historical matrices, `qwen/qwen3-8b`. The final report emphasizes saved complete runs and avoids comparing incomplete reports as if they were full evaluations.

4.2. Metric Evolution and Reproducibility

The contest metric changed during the submission period. Early work optimized raw flattened Micro-F1 and per-instance task score patterns. Later documentation added explicit null hallucination penalties, list matching details, gap-closed ranking relative to baseline, multi-model aggregation, and soft average token/time budgets. The repository includes tests that pin important parts of the documented evaluator, including null handling, date rigidity, and corpus-level Micro-F1, but the final metric version was not as thoroughly validated before the paper deadline as earlier metrics. The current snapshot additionally includes final Gemma 4 E4B and Qwen3 14B no-think self-evaluation reports over `data/test` with 145 instances. This creates a reproducibility limitation: historical experiments remain useful for development insight but should not be interpreted as final leaderboard-equivalent scores, and the final table should be read as local self-evaluation evidence rather than an official ranking.

We therefore report results in phases. Baseline analyses describe the early model/task landscape. Historical strategy matrices show development trends across pipelines. Grounded-adaptive v6.1 results summarize a late optimization pass under the then-current evaluator. Bad-instance re-benchmarks stress-test candidate pipelines on the curated 74-instance set. The final self-evaluation reports Micro-F1 and gap-closed over the baseline for the actual submitted pipelines on the two requested models. The code snapshot referenced below is commit `39e997cc3a88b514f203c75259a3087310cfac4d`.

5. Results

5.1. Baseline Cross-Model Analysis

The baseline analysis over starter and development sets showed that model choice mattered, but the dominant difficulty was task family. Table 4 summarizes the saved aggregate baseline metrics. As indicated in Table 2, these early numbers are not directly comparable with later final-pipeline

Table 2
Result provenance and comparability. "Final-compatible" means the report uses the later flattened Micro-F1 style saved in the repository, not that it is an official final leaderboard score.

Phase	Dataset / instances / models	Report generation	Interpretation
Baseline cross-model	starter/dev; 40/149; Qwen, Llama, Gemma	Early baseline analysis report	Not directly comparable to final runs; useful for task difficulty and model sensitivity.
Historical matrix	starter/dev/hard; mixed saved splits; Gemma, Llama, Qwen	Broad matrix report generation	Comparable within table, but pipeline names differ from the final submission.
Grounded adaptive v6.1	starter/dev/hard; 9 model–split cells	Late grounded-adaptive development summary	Development evidence for `grounded`; not the exact final entry.
Bad-instance re-benchmark	curated bad-case; 74; Qwen 1.7B, plus selected two-model ReAct runs	Aggregate server campaign over proposal variants	Controlled within campaign only; not official leaderboard evidence.
Final self-evaluation	`data/test`; 145; Gemma 4 E4B and Qwen3 14B, no-think	Saved final-pipeline and baseline reports from the 20260630 Gemma and 20260701 Qwen report directories	Final-compatible local evidence for both requested models; not an official leaderboard run.

Table 3
Reproducibility details for the saved evidence reported in this paper.

Item	Value
Code snapshot	Git commit `39e997c`; manuscript files were untracked during preparation unless committed after this report.
Pipeline selectors	`combo_guard_react_repair_ref`, `grounded`, `combo_guard_list_spacy_ref`.
Models in reported runs	`qwen/qwen3-1.7b`, `qwen3-14b`, `google/gemma-4-e2b`, `google/gemma-4-e4b`, `llama-3.2-3b-instruct`; historical matrix also includes `qwen/qwen3-8b`.
Decoding	Temperature 0 in the final pipelines; no sampling seed is used because runs are deterministic at request level but backend implementations may still vary.
Budgets and timeouts	Soft averages: 32K tokens and 60s. `grounded`: 50s call timeout, 38s recovery budget, 2048 evidence tokens, 4096 extraction tokens.
Main dependencies	Python 3.13+; `openai` 1.50+; `json-repair` 0.59.10+ with schema extra; spaCy 3.8.14+; `fastembed` 0.3+.
External models/tools	spaCy `es_core_news_lg`; fastembed `BAAI/bge-small-en-v1.5`.
Final self-evaluation	Gemma 4 E4B and Qwen3 14B no-think results are available for `data/test`; values are local saved evidence rather than official leaderboard outputs.

Table 4
Baseline starter/development metrics from the repository cross-model analysis; see Table 2 for evaluator/report provenance.

Model	Split	P	R	F1
Qwen 1.7B	starter	0.1508	0.1397	0.1450
Qwen 1.7B	dev	0.0632	0.0605	0.0618
Llama 3.2 3B	starter	0.1382	0.1234	0.1304
Llama 3.2 3B	dev	0.0599	0.0554	0.0576
Gemma 4B	starter	0.1598	0.1453	0.1522
Gemma 4B	dev	0.0685	0.0614	0.0648

measurements. The starter set was substantially easier than the development set for all three models. Gemma was the strongest baseline on both splits, but all models had high bad-solution rates on development instances.

The baseline error analysis identified three recurring weaknesses. Attribute filling was harder than mixed reasoning in the saved classification, especially for long records with many nullable fields. Named entity listing produced label confusions and partial lists. Text-span QA suffered from paraphrase mismatches, where semantically close outputs failed rigid or lexical scoring. These findings motivated

Table 5
Historical complete-run matrix used during development. Values are Micro-F1; see Table 2 for comparability limits.

Dataset	Model	Baseline	strict_fast	grounded_adaptive	toon_qa
starter	Gemma 4B	0.6999	0.7252	0.7356	0.7070
starter	Llama 3.2 3B	0.6314	0.6695	0.6747	0.6996
starter	Qwen 1.7B	0.6752	0.7261	0.7285	0.7546
dev	Gemma 4B	0.6841	0.7111	0.7118	0.6356
dev	Llama 3.2 3B	0.6471	0.6802	0.6586	0.6583
dev	Qwen 1.7B	0.6499	0.7209	0.7250	0.7412
hard	Gemma 4B	0.5582	0.7246	0.7087	0.6648
hard	Llama 3.2 3B	0.4572	0.7098	0.7093	0.6786
hard	Qwen 1.7B	0.4844	0.7406	0.7303	0.6922

schema verbalization, list-recall prompts, span preservation, and stricter null handling.

5.2. Historical Pipeline Matrix

The broad historical matrix compared `strict_fast`, `grounded_adaptive`, and `toon_qa` across starter, development, and hard datasets. These are not the final three pipeline names, but they explain the design path. Table 5 reports representative complete runs from one report generation; Table 2 marks them as development evidence rather than final-submission results. `grounded_adaptive` was the best general starter default, `strict_fast` was strongest on most hard-set model combinations, and `toon_qa` was highly competitive for Qwen on starter/dev but less stable elsewhere.

The matrix suggests that no single method dominated every model and split. The strongest pattern was that stricter schema handling and repair substantially improved hard-set performance over the baseline, while more elaborate reasoning or alternative intermediate representations could help selected model/task pairs but introduced latency and stability costs.

5.3. Grounded Adaptive Evolution

The grounded-adaptive v6.1 summary compared a late branch against v6 Branch A across nine model/dataset cells. The 9-cell average improved from 0.7064 to 0.7105. Dataset-level averages for v6.1 were 0.7333 on starter, 0.6971 on dev, and 0.7009 on hard, with average token use per instance of 3257.8, 6572.8, and 5931.5, respectively. The changes that mattered most were conservative named-entity repair, deterministic QA sentence/fragment routing, drug-description stabilization, recipe-time nulling when unsupported, and media-review cleanup.

These results influenced the final `grounded` pipeline, especially its schema digest, optional evidence pass, time-budget guard, long-text retrieval, and targeted recovery logic. However, the exact v6.1 pipeline is not identical to the final `OfficialParticipant` entry; we treat it as development evidence rather than a final submission result.

5.4. Bad-Instance Re-Benchmark

The 74-instance bad-case set was designed to stress cases that multiple baselines handled poorly. A large re-benchmark over 21 implementations on Qwen 1.7B ranked several simple prompt/repair variants near the top. The best five F1 scores were 0.6794 for `idea_basicagent_react_lite`, 0.6790 for `idea_basicagent_text_only`, 0.6784 for `mixed_f`, 0.6729 for `mixed_g`, and 0.6599 for `idea_basicagent_toon_soft`; the baseline was 0.6197. This phase showed that many gains came from disciplined prompting and robust parsing rather than expensive agent loops.

A later ReAct-family comparison on the same bad-case set and two models produced a different view: `react_checklist_guard_ref` had the best mean F1 at 0.7061, followed by `react_field_qa_ref`

at 0.7045, while one-call guard references remained highly efficient. The consensus variant had similar F1 but consumed far more tokens and time. This result motivated the final guard-first repair design: validate and repair only when specific issue kinds justify the extra call.

5.5. Final Self-Evaluation for the Submitted Pipelines

Table 6 reports the final no-think self-evaluation saved for the actual submitted pipeline selectors on the two requested models. These runs use `data/test` with 145 instances and follow the repository self-evaluation protocol: Micro-F1 is reported directly, and gap-closed is computed as $\max(0, (F1_{\text{system}} - F1_{\text{baseline}})/(1 - F1_{\text{baseline}}))$. For Qwen, the table uses the patched report files where present after retry merging. Table 2 should be used when interpreting comparability with earlier development results.

Table 6
Final no-think self-evaluation on `data/test` ($n = 145$). Values are local saved evidence from `results/gemma4e4b_gold_fast_nothink_live_20260630_151100` and `results/qwen3_14b_fast_nothink_live_20260701`; see Table 2.

Model	Pipeline	F1	Gap	P	R	Avg s	Avg tok.	Calls
Gemma E4B	`baseline`	0.7128	–	0.7873	0.6512	17.26	3735	127
Gemma E4B	`combo_guard_react_repair_ref`	0.8091	0.3353	0.8147	0.8035	31.27	6140	209
Gemma E4B	`combo_guard_list_spacy_ref`	0.7671	0.1891	0.8082	0.7300	22.16	6729	144
Gemma E4B	`grounded`	0.4555	0.0000	0.6641	0.3467	28.74	5357	190
Qwen 14B	`baseline`	0.6839	–	0.7801	0.6088	24.15	3688	127
Qwen 14B	`combo_guard_react_repair_ref`	0.8003	0.3681	0.8091	0.7916	32.28	5240	200
Qwen 14B	`combo_guard_list_spacy_ref`	0.7957	0.3538	0.8067	0.7851	36.13	6313	145
Qwen 14B	`grounded`	0.7279	0.1390	0.7653	0.6939	35.32	4688	189

The final self-evaluation supports three conclusions. First, guard-first repair is the strongest submitted pipeline on both models, closing 33.5% of the Gemma baseline-to-perfect gap and 36.8% of the Qwen gap, for a two-model average of 35.2%. Second, spaCy evidence improves over both baselines and is especially competitive on Qwen, with a two-model average gap-closed score of 27.1%. Third, `grounded` remains model-sensitive: it underperforms the baseline in the Gemma no-think run but closes 13.9% of the Qwen gap, averaging 7.0% over the two models.

5.6. Efficiency

Efficiency was a constant constraint. Historical results show that strict/guarded single-call pipelines generally had the lowest wall time, while evidence-first and TOON/QA variants consumed more time and tokens. In the final no-think self-evaluation, the baseline averaged 17.26 seconds and 3735 tokens per Gemma instance, and 24.15 seconds and 3688 tokens per Qwen instance. `combo_guard_react_repair_ref` averaged 31–32 seconds per instance and used 209 Gemma calls and 200 Qwen calls over 145 tasks, reflecting selective escalation beyond the first call. `combo_guard_list_spacy_ref` kept call counts low but used the largest prompts, averaging 6729 Gemma tokens and 6313 Qwen tokens per instance. `grounded` used 190 Gemma calls and 189 Qwen calls. No final local report exceeded the 32K target or 64K soft token caps; the maximum observed total was 18,612 tokens, so these saved self-evaluations did not reproduce the context-overflow issue reported by the organizers for their formal environment.

5.7. Error Analysis

Across phases, the recurring error categories were stable:

- **Span or paraphrase mismatch:** systems returned semantically close but lexically different values, hurting rigid or partly lexical scoring.

Table 7
Task-level mechanical diff frequencies on saved bad-case reports. Counts are tasks affected out of 74.

Pipeline	Model	Miss.	Extra	Null+	Null−	List	Scalar
Guard repair	Qwen 1.7B	33	27	23	15	50	62
Guard repair	Gemma 4B	36	25	15	32	48	61
spaCy evidence	Qwen 1.7B	55	45	27	12	50	45
spaCy evidence	Gemma 4B	36	32	14	32	51	56

- **List underfill and overfill:** models missed explicit list items or added unsupported ones, especially for ingredients, media pros/cons, cast lists, and entity extraction.
- **Null handling:** models hallucinated non-null values for absent fields or omitted required nullable slots.
- **Entity label confusion:** named entity tasks confused person, organization, miscellaneous, event, and domain-specific labels.
- **Schema-shape drift:** evidence-rich prompts occasionally caused the model to output diagnostic/evidence objects instead of the target schema.
- **Rigid normalization:** dates, numeric strings, Spanish decimals, enum casing, and legal/cultural codes required exactness that models did not always preserve.

To make this qualitative analysis more concrete, Table 7 reports task-level mechanical diff frequencies over the four saved bad-case final-pipeline reports. These are not hand-audited causal labels: a task can appear in several columns, and nested arrays can make one underlying failure surface as list, missing, extra, and scalar changes. Still, the pattern is useful. The spaCy-Qwen run has many more tasks with missing and extra fields, supporting the diagnosis that the evidence representation sometimes caused schema-shape drift.

These errors explain the final design. Guarding and repair target shape, enum, and null errors. Grounding targets hallucinated values and list recall. spaCy evidence targets candidate-span discovery, but its failures show that evidence must be presented in a way that cannot be mistaken for the output schema.

6. Discussion

The CodeStrange experiments support a conservative view of schema-guided extraction with small models. More model calls do not automatically improve results. Consensus and multi-pass reasoning can increase cost faster than accuracy, and complex evidence prompts can break smaller models. The most reliable gains came not from deeper reasoning loops, but from conservative schema control: clear schema verbalization, JSON-only prompting, deterministic cleanup, explicit null discipline, and targeted repair triggered by concrete validation issues.

At the same time, the best method depends on the model and reasoning mode. In the final no-think self-evaluation, guard-first repair is the strongest saved run on both Gemma 4 E4B and Qwen3 14B. spaCy evidence also improves over both baselines and comes close to guard repair on Qwen, but historical hard-case results still show that a similar evidence representation failed badly on Qwen 1.7B because the model copied or transformed the evidence schema into the answer. Llama was more sensitive to latency and output complexity in earlier runs. This model dependence justifies GenSIE's multi-model design and suggests that future systems should route by observable schema/task difficulty and output quality signals rather than by hard-coded model names.

The metric evolution also shaped development. When raw Micro-F1 was the main local target, several strategies optimized key coverage and task-score deltas. Later gap-closed ranking changed the interpretation of gains across models. The null hallucination penalty and list matching rules made conservative grounding and list handling more important. The final no-think self-evaluations are compatible with this later local protocol, but they remain self-reported local evidence rather than official

leaderboard outputs. The safest scientific conclusion is therefore not that one pipeline is definitively best across every execution environment, but that validation-aware, schema-aware extraction is the most robust design pattern observed in the repository.

7. Conclusions, Future Challenges, and Recommendations

CodeStrange submitted a three-pipeline GenSIE system built around inference-time control rather than model fine-tuning. The guard-first ReAct repair pipeline provides robust validation and targeted correction. The grounded pipeline provides the broadest general-purpose architecture, combining schema digests, evidence cues, retrieval for long texts, and recovery. The spaCy evidence pipeline shows that classical NLP can improve field grounding and recall for some models, but also exposes a serious output-shape risk when evidence is too structurally similar to the requested answer. In the final Gemma 4 E4B and Qwen3 14B no-think self-evaluations, guard-first repair is the best submitted pipeline, averaging 35.2% gap-closed; spaCy evidence averages 27.1%, and `grounded` averages 7.0%. The main lesson is that, for small and medium language models in schema-guided Spanish information extraction, robust inference-time control matters more reliably than complex reasoning loops.

Future work should focus on four challenges. First, evidence must be represented in a way that helps extraction without becoming a template for the model's output. Second, validators should become more semantic: current deterministic checks can identify schema and obvious grounding issues but cannot fully judge paraphrase quality or evidence support. Third, metric-stable evaluation is essential; when shared-task metrics change late, participants need enough time to rerun final candidates. Fourth, model-agnostic routing remains unsolved: even when the same ranking held for the two final no-think self-evaluations, historical runs show that smaller and alternative models can respond very differently to evidence and repair.

We recommend future GenSIE systems use a conservative first pass with schema verbalization and JSON-only instructions, deterministic cleanup before any second LLM call, explicit null and enum rules, lightweight evidence only for fields where it is likely to help, and repair loops gated by concrete issue types. For challenge organization, we recommend freezing metric definitions earlier, publishing evaluator version identifiers in reports, and providing a small final-metric validation window before the paper deadline. These steps would improve reproducibility without reducing the task's emphasis on robust, grounded, schema-adaptive extraction.

Declaration on Generative AI

During the preparation of this work, the authors used OpenAI ChatGPT/Codex, Anthropic Claude Code, and OpenCode as AI-assisted writing and coding tools. The tools were used to inspect repository artifacts, summarize experimental reports, draft and revise manuscript text, check consistency with CEUR-WS formatting requirements, and assist with LaTeX compilation diagnostics. The authors reviewed, edited, and approved the final manuscript and take full responsibility for the scientific content, claims, references, results, and conclusions. No generative AI tool is listed as an author.

Acknowledgments

We thank the GenSIE organizers for releasing the starter kit, documentation, evaluator, and development data that made local experimentation possible.

References

[1] 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, p. to appear.

[2] 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, p. to appear.

[3] Y. Almeida Cruz, S. Estévez Velarde, A. Piad Morffis, I. Espinosa Zaragoza, M. Miró Maestre, A. Pérez Montero, L. Sevilla Requena, E. Estevanell Valladares, GenSIE: General-purpose schema-guided information extraction. task proposal for IberLEF 2026, 2026. Task proposal and project materials.

[4] D. Xu, W. Chen, W. Peng, C. Zhang, T. Xu, X. Zhao, X. Wu, Y. Zheng, Y. Wang, E. Chen, Large language models for generative information extraction: a survey, Frontiers of Computer Science 18 (2024). doi:10.1007/s11704-024-40555-y.

[5] Z. Zhang, W. You, T. Wu, X. Wang, J. Li, M. Zhang, A survey of generative information extraction, in: Proceedings of the 31st International Conference on Computational Linguistics, Association for Computational Linguistics, Abu Dhabi, UAE, 2025, pp. 4840–4870. URL: https://aclanthology.org/2025.coling-main.324/.

[6] R. Han, T. Peng, C. Yang, B. Wang, L. Liu, X. Wan, An empirical study on information extraction using large language models, 2024. arXiv:2305.14450.

[7] S. Geng, M. Josifoski, M. Peyrard, R. West, Generating structured outputs from language models: Benchmark and studies, 2025. arXiv:2501.10868.

[8] Y. Dong, C. F. Ruan, Y. Cai, R. Lai, Z. Xu, Y. Zhao, T. Chen, XGrammar: Flexible and efficient structured generation engine for large language models, 2024. doi:10.48550/arXiv.2411.15100. arXiv:2411.15100, MLSys 2025.

[9] Z. Shen, D. Y.-B. Wang, S. S. Mishra, Z. Xu, Y. Teng, H. Ding, SLOT: Structuring the output of large language models, in: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, Association for Computational Linguistics, Suzhou, China, 2025, pp. 472–491. URL: https://aclanthology.org/2025.emnlp-industry.32/. doi:10.18653/v1/2025.emnlp-industry.32.

[10] S. Baccianella, json_repair: A Python package to repair broken JSON strings, including malformed JSON commonly produced by LLMs, https://github.com/mangiucugna/json_repair, 2026. Software version 0.61.0; upstream `CITATION.cff` metadata.

[11] A. H. M. R. Karim, Ö. Uzuner, Retrieval-augmented large language models for schema-constrained clinical information extraction, 2026. doi:10.48550/arXiv.2605.15467. arXiv:2605.15467.

[12] Y. Luo, X. Ru, K. Liu, L. Yuan, M. Sun, N. Zhang, L. Liang, Z. Zhang, J. Zhou, L. Wei, D. Zheng, H. Wang, H. Chen, OneKE: A dockerized schema-guided LLM agent-based knowledge extraction system, 2024. doi:10.48550/arXiv.2412.20005. arXiv:2412.20005, WWW 2025 Demonstration.

[13] Explosion AI, spaCy: Industrial-strength natural language processing in Python, https://spacy.io/, 2026. Accessed: 2026-06-16.

[14] Explosion AI, EntityRuler: spaCy API documentation, https://spacy.io/api/entityruler, 2026. Accessed: 2026-06-16.

[15] A. Shrimal, A. Jain, S. Chowdhury, P. Yenigalla, PARSE: LLM driven schema optimization for reliable entity extraction, in: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, Association for Computational Linguistics, Suzhou, China, 2025, pp. 2749–2763. URL: https://aclanthology.org/2025.emnlp-industry.184/. doi:10.18653/v1/2025.emnlp-industry.184.

[16] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, 2022. arXiv:2203.11171.