

Constructing Knowledge Graphs from Agentic AI Frameworks

Naura Ranantya Kinanti¹, Ravie Arjun Nadhief¹, Kabul Kurniawan^{1*}, Guntur Budi Herwanto¹, Elmar Kiesling⁴, and Andreas Ekelhart^{2,3}

¹ Department of Computer Science and Electronics, Universitas Gadjah Mada, Indonesia

² SBA Research, Vienna, Austria

³ University of Vienna, Vienna, Austria

⁴ Vienna University of Economics and Business, Austria

Abstract. Modern agentic AI frameworks such as AutoGen, CrewAI, LangGraph, and Mastra AI encode agent roles, tasks, tools, and workflow logic directly in framework-specific code, which makes these designs difficult to inspect, compare, reuse, or audit. We present an LLM-guided pipeline that maps heterogeneous agentic workflows to knowledge graphs compliant with the Agentic AI Ontology (AgentO), a framework-independent, RDF/OWL-based vocabulary for agents, tasks, goals, tools, and workflow patterns. The pipeline concatenates each pattern’s source files as context, applies an ontology-grounded prompt to a large language model, and validates the resulting Turtle output before publishing it to a knowledge graph. Across four frameworks, we assess syntactic validity, semantic completeness, and semantic conformance, and compare prompting strategies and backend models. The pipeline produces valid knowledge graphs, with relation extraction remaining the main bottleneck. Prompt design, not model choice, is the dominant lever on extraction quality.

Keywords: Agentic AI · Knowledge Graphs · Ontology Population · Large Language Models

1 Introduction

Agentic AI systems – defined by autonomy, goal-directed behaviour, and coordinated action across multiple agents [1] – are increasingly trusted to orchestrate complex, multi-step workflows in domains as varied as cybersecurity [2], financial services [3], industrial supply chains [4], healthcare [5], and scientific discovery [6]. Frameworks such as AutoGen [7], CrewAI [8], LangGraph [9], and Mastra AI [10] give agents roles, goals, and tools to plan and carry out these workflows, but adoption has outpaced standardization: each framework encodes agent logic and coordination in its own ad-hoc code and configuration, with no shared, machine-readable representation underneath. This tight coupling makes

* Corresponding author: kabul.kurniawan@ugm.ac.id

agentic systems difficult to inspect, compare across frameworks, reuse, and audit – shortcomings that matter more as such systems enter regulated domains [11].

Knowledge graphs and ontologies address exactly this gap: they give entities and relations an explicit, queryable, machine-readable structure [12]. The Agentic AI Ontology (AgentO) [18] was designed for this purpose, formally capturing agents, tasks, goals, tools, resources, and workflow patterns independently of any particular framework. But hand-authoring AgentO instances for the large body of existing agentic implementations does not scale. This paper addresses that gap: *can existing, framework-specific agentic AI implementations be automatically lifted into ontology-compliant knowledge graphs, and how reliable is such a construction process?*

We present and evaluate an LLM-guided pipeline that concatenates a pattern’s source files as context, applies an ontology-grounded prompt treating AgentO as a fixed schema, and validates the resulting Turtle output. LLMs are well suited here because source code varies in structure, naming, and completeness across frameworks, and carries semantically relevant content in natural-language fields that rule-based parsers cannot generalize across. Anchoring generation to the ontology schema constrains hallucination risk [19] while allowing the LLM to bridge framework-specific idioms. We evaluate the pipeline on 50 existing agentic patterns spanning four frameworks across four axes: (i) *syntactic validity*; (ii) *semantic completeness* via precision, recall, and F_1 under a six-variant, two-stage prompt ablation; (iii) *semantic conformance* via SHACL; and (iv) sensitivity to backend LLM choice.

The remainder of this paper is organized as follows: Section 2 surveys related work; Section 3 recapitulates AgentO; Section 4 describes the pipeline; Section 5 reports the evaluation; Section 6 discusses results and threats to validity; and Section 8 concludes.

2 Related Work

Semantic Web and Agentic AI Agent-based mediation across heterogeneous environments was part of the Semantic Web vision from the outset [21], extending to semantic service annotation (WSMO, WSDL-S) before machine learning absorbed much of its intended role [22]. What endured is the knowledge graph as a practical mechanism for machine-interpretable entity and relation representation [12]. Modern agentic AI systems – LLM agents that plan, invoke tools, and coordinate beyond single-pass inference [23] – are served by frameworks such as AutoGen, CrewAI, and LangGraph [7,8,9], none of which provide semantic portability across framework boundaries. AgentO [18] takes up the original agent-oriented vision under a durable KG representation, offering a framework-independent vocabulary for agents, tasks, goals, tools, and workflow patterns.

LLM-Based Knowledge Graph Construction and Ontology Population LLMs are increasingly used to construct and populate knowledge graphs from text [13]. Ontology learning approaches such as LLMs4OL induce ontology structure in a zero-

shot setting over open vocabularies [14]. SPIRES/OntoGPT performs schema-constrained, zero-shot extraction of instance data from natural-language text against a fixed ontology [15]. Recent work further shows that LLMs can produce high-quality KGs from unstructured and semi-structured text when guided by ontology constraints [16,17]. Our pipeline shares this schema-constrained, zero-shot design but differs in extraction target: rather than extracting from natural-language prose, it populates AgentO from heterogeneous agentic AI *source code*.

Gap Where knowledge graphs have previously met agentic systems, the graph has typically served as a *runtime* resource that an agent consults while reasoning or retrieving information, rather than as a specification of the agent system itself. Earlier work on knowledge-graph-enhanced RAG for cybersecurity analysis [24] is an instance of this pattern: the graph is queried at inference time, not used to describe the agents doing the querying. At the same time, the multi-agent LLM frameworks discussed above seldom expose their own agents, tasks, and coordination logic through a shared, machine-readable vocabulary, leaving most of this ecosystem unqueryable and incomparable across framework boundaries. This paper targets that gap directly: it treats agentic AI source code as an extraction corpus and AgentO as a fixed target schema, and systematically evaluates how reliably existing, framework-specific implementations can be lifted into a single, declarative, cross-framework knowledge graph.

3 The AgentO Ontology

The construction pipeline targets the Agentic AI Ontology (AgentO) [18], an OWL/RDF vocabulary built bottom-up from a survey of widely used agentic AI frameworks following standard ontology engineering practice [25]: identifying recurring components across CrewAI, AutoGen, LangGraph, and Mastro AI implementations (*Agent*, *Task*, *Tool*, ...), modelling the relationships and workflow patterns (sequential, parallel, nested) that connect them, and formalizing the result as OWL classes and properties. AgentO reuses established Semantic Web vocabulary where it fits – `prov:Agent` from PROV-O [26], `pplan:Plan/pplan:Step` from P-Plan [27], and `beam:Resource` from the Boxology/BEAM notation [28] – and functions as a fixed, framework-independent target schema for extraction completeness. Because it was distilled from the same four frameworks it is here evaluated against, this pairing introduces a construct-*validity* concern – the ontology may be better suited to represent the four studied frameworks than agentic systems in general – which Section 7 addresses.

Table 1 lists the core classes the pipeline instantiates (a `:Team` of `:LLMAgent` individuals executes a `:WorkflowPattern` of ordered `:WorkflowStep` individuals, each tied to a `:Task` that may draw on a `:Tool` and a `:Resource`); the complete class and property listing, the ontology file, and a diagram of how these classes relate are available in the project’s public repository.⁵

⁵ <https://github.com/raviearjun/agentic-generator-paper>

Table 1. Core AgentO classes instantiated during construction (excerpt; the full listing is available in the project repository).

Class	Role in a construction instance
<code>:Team</code>	Coordinated group of agents pursuing a shared goal
<code>:LLMAgent</code>	Autonomous agent based on a language model
<code>:Task</code>	Concrete activity performed by an agent
<code>:Tool</code> / <code>:Resource</code>	Instrument used by, or asset consumed by, a task
<code>:WorkflowPattern</code> / <code>:WorkflowStep</code>	Reusable workflow structure and its steps
<code>:Prompt</code> / <code>:Config</code>	Agent instructions and configuration settings

4 LLM-Guided Knowledge Graph Construction

This section describes the pipeline that operationalizes AgentO by lifting existing, framework-specific agentic workflows into ontology-compliant knowledge graphs. Unlike the inverse, KG-to-code compilation direction, which is deterministic and template-driven, construction is inherently an open-ended extraction problem: source code varies in structure, naming conventions, and completeness across frameworks, and the mapping from framework-specific constructs to ontology terms is neither one-to-one nor predictable from a fixed grammar. A conventional, rule-based or AST-based extractor would require a hand-crafted parser per framework, and could not generalize to the implicit conventions, natural-language fields (agent roles, task descriptions, goal text), and configuration idioms that carry most of the semantically relevant content. An LLM guided by a fixed ontology schema handles these variations without per-framework engineering; the schema closes the vocabulary, and the LLM supplies the cross-framework generalization. The pipeline therefore relies on an LLM as the extraction engine, constrained by the AgentO schema. The pipeline consists of four stages, described in turn and summarized in Figure 1.

1. Source Pattern Collection We collected existing agentic workflows from the official public example repositories of each framework: 6 from AutoGen, 16 from CrewAI, 8 from LangGraph, and 20 from Mastra AI (50 patterns in total). Patterns were selected to represent the range of workflow types each framework supports (single-agent, multi-agent, sequential, parallel, tool-using, and memory-augmented), prioritizing examples with self-contained source files and excluding pure SDK wrappers or frontend-only code that contain no genuine agentic implementation. The project repository lists each pattern with its framework, tool usage, and workflow type. These patterns are hard-coded configurations within Python or TypeScript files – for example, a CrewAI agent is instantiated with a `role`, `goal`, `backstory`, and a list of `tools`, and is assigned one or more `Task` objects with a `description` and `expected_output`, which are then composed into a `Crew` that coordinates execution.

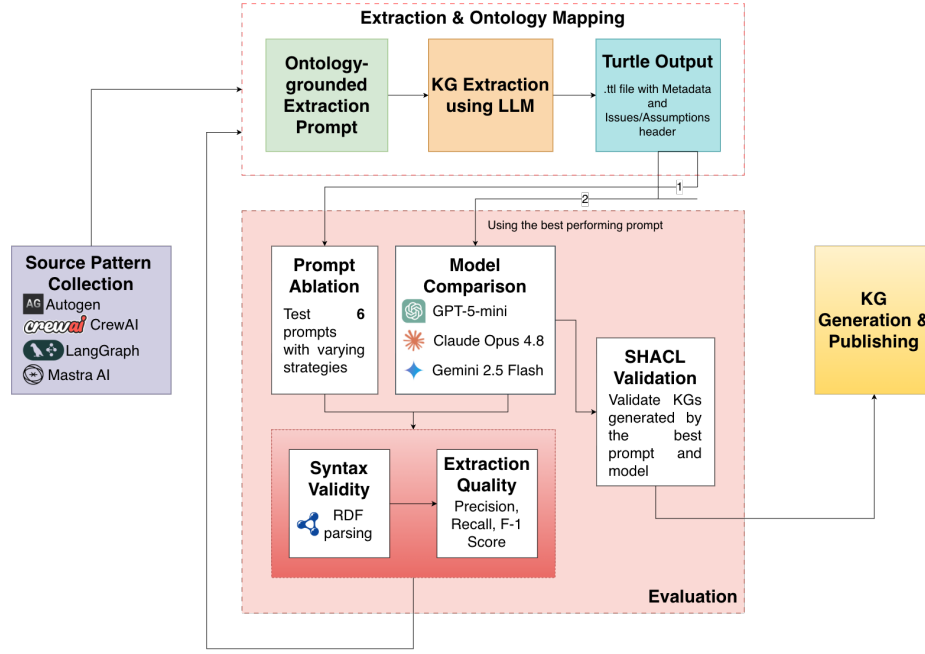


Fig. 1. Overview of the LLM-guided knowledge graph construction pipeline: Source Pattern Collection, Extraction and Ontology Mapping, Evaluation, and KG Generation and Publishing.

2. Pattern Extraction and Ontology Mapping For each pattern, every source file belonging to that pattern is concatenated into a single context and passed, together with an ontology-grounded prompt, to an LLM. The prompt treats the ontology as a closed, fixed schema: the model is instructed to instantiate *only* existing AgentO classes and properties, to preserve source information as literals or relations rather than paraphrasing it away, and to record any concept it cannot express in the current schema in a structured *Issues/Assumptions* header rather than inventing new vocabulary. This constraint is what keeps the resulting KG’s vocabulary closed and lets it be evaluated against a fixed gold standard (Section 5.1). All primary extractions use GPT-5-mini via API access; Section 5.2 additionally compares this backend against two alternative LLMs. Listing 1.1 shows a representative excerpt of the generated Turtle for the CrewAI *Game Builder Crew* pattern (one of the 41 SHACL-evaluable files summarized in Table 5), illustrating the `:Team/:LLMagent` structure the pipeline instantiates from source code.

```

1 :team_game_builder rdf:type :Team ;
2   dct:terms:title "GameBuilderCrew" ;
3   :hasAgentMember :agent_senior_engineer , :agent_qa_engineer ,
4   :agent_chief_qa_engineer ;
5   :hasWorkflowPattern :wp_sequential ;
6   :hasTeamGoal :goal_team .

```

```

7 :agent_chief_qa_engineer rdf:type :LLMAgent ;
8 :agentID "chief_qa_engineer_agent" ;
9 :agentRole "Chief Software Quality Control Engineer" ;
10 :agentPrompt :prompt_chief_qa_engineer ;
11 :useLanguageModel :lm_gpt4o ;
12 :hasAgentGoal :goal_chief_qa_engineer ;
13 :agentToolUsage :tool_openai_api , :tool_serper .
14

```

Listing 1.1. Representative excerpt of generated Turtle for the CrewAI *Game Builder Crew* pattern.

3. *Evaluation and Publishing* Each generated Turtle file is parsed via `rdflib` for syntax validity, scored for semantic completeness against a per-framework gold standard, and validated with `pySHACL`. Validated output across all patterns is compiled into a single RDF graph hosted in a triple store and made publicly queryable via SPARQL, turning each pattern into a queryable instance of a shared vocabulary.

5 Evaluation

This section presents a systematic evaluation of the KG construction pipeline. The evaluation is structured around three evaluation questions. **EQ1:** How reliably does the extraction pipeline produce syntactically valid RDF/OWL output across frameworks and models? **EQ2:** Which prompting strategies most effectively guide an LLM in populating the KG? **EQ3:** How do different LLM backends compare as extraction engines? Beyond these three questions, we additionally assess whether the extracted knowledge graph conforms semantically to the ontology’s constraints, beyond mere completeness, via SHACL validation.

5.1 Setup and Metrics

The evaluation dataset comprises the 50 patterns described in Section 4 (6 AutoGen, 16 CrewAI, 8 LangGraph, 20 Mastra AI), evaluated in two parts: a *prompt ablation* evaluates six prompt variants, P0–P5⁶, in two stages – Stage 1 runs GPT-5-mini on 5 controlled examples per framework (20 total, identical across prompts), and Stage 2 reruns the three best Stage-1 prompts on the full per-framework set (6–20 examples) for a more reliable estimate (P0 is the baseline; P1 strips it to the core objective; P2 adds structured extraction rules; P3 adds two-phase, analyze-then-generate reasoning; P4 adds a self-validation checklist; P5 adds framework-specific mapping guidance on top of P2); and a *model comparison* evaluates GPT-5-mini, Claude Opus 4.8, and Gemini 2.5 Flash on identical prompts and source code, using the best prompt and framework from the ablation (CrewAI, P5) to isolate the effect of the model itself.

⁶ The complete prompt texts are provided in the project’s public repository, https://github.com/raviearjun/agentic-generator-paper/tree/main/script_to_kg/prompts.

Three metrics are reported. *Syntax validity*: a file is valid if it parses without error as an RDF graph via `rdflib`; only valid files proceed to semantic scoring. *Extraction quality*: for each level $l \in \{cls, obj, data\}$, a generated file G is compared to a per-framework gold standard G_f ⁷. False positives are defined as terms present in G but absent from the full ontology schema, not merely absent from G_f : valid ontology terms outside the gold set are not penalized, as they represent legitimate but non-mandatory extractions. – *True Positives* are G_f terms present in G , *False Negatives* are G_f terms missing from G , and *False Positives* are terms in G undefined in the ontology (valid terms outside G_f but within the schema are not penalized, as legitimate extrapolations); the three F_1 scores are macro-averaged:

$$\text{Macro-}F_1 = \frac{F_{1,cls} + F_{1,obj} + F_{1,data}}{3}, \quad (1)$$

rewarding completeness and penalizing schema invention. *Semantic conformance*: since precision/recall/ F_1 check only whether expected terms are *present*, not whether they are used *correctly*, every file is additionally validated with pySHACL against a shapes⁸ auto-derived from the ontology’s domain, range, and datatype declarations (framework-agnostic, applied uniformly). Eight minimum-cardinality constraints (e.g., every `:LLMAgent` should have an `:agentID`) are encoded as `sh:Warning` rather than `sh:Violation` to separate soft completeness gaps from hard errors.

5.2 Extraction Evaluation Results

Syntax Validity (EQ1) Of the 265 generated files across the full dataset (all prompt variants, models, and frameworks combined), 254 are syntax-valid (95.8%). All 22 Claude files⁹ and all 33 Gemini files¹⁰ are valid (100%); 11 of 210 GPT-5-mini files¹¹ fail (94.8% validity), with 2 failures out of 50 files on prompt P5. By

⁷ G_f is a manually curated set of AgentO classes, object properties, and data properties that a correct extraction of a *typical* pattern from framework f should contain. One annotator (a co-author with knowledge of both AgentO and the target frameworks) examined representative patterns per framework and identified the AgentO terms any well-formed extraction should include (e.g., every CrewAI pattern has agents, tasks, a workflow, and config entries, so the gold set includes `:LLMAgent`, `:Task`, `:WorkflowPattern`, and their connecting properties). Per-example overrides were added where a pattern legitimately deviates from the framework baseline (e.g., a single-agent example has no `:Team`; an example without external tools has no `:Tool` or `:agentToolUsage`)

⁸ The complete SHACL shape file is available at https://github.com/raviearjun/agentic-generator-paper/blob/main/script_to_kg/evaluation/shacl/core.shapes.ttl

⁹ 16 CrewAI files using prompt P5, 2 CrewAI files using P0, 1 CrewAI file each for prompts P1-P4 generated during experimentation

¹⁰ 16 CrewAI files using prompt P5; 3 CrewAI files each for P0, P1, and P3; 4 CrewAI files each for P2 and P4 generated during experimentation

¹¹ Files generated from all experiments

framework, AutoGen reaches 100% validity, CrewAI 99.2%, LangGraph 94.9%, and Mastra AI 89.3%. All 11 invalid files share a single error class – a multi-line string literal delimited with a single-quoted rather than triple-quoted string, causing the parser to fail at the first embedded newline – distributed across CrewAI, LangGraph, and Mastra AI examples and multiple prompts, indicating a stochastic generation artefact rather than a framework- or example-specific failure.

Prompt Ablation (EQ2) Table 2 reports Stage-1 results (5 examples per framework, controlled). P4 (self-validation checklist) achieves the highest controlled macro- F_1 (0.959), closely followed by P2 (0.957) and P5 (0.951). P1 scores below the P0 baseline (0.901 versus 0.929), indicating that extraction directives alone, without format or content guidance, narrow the model’s scope; P2 recovers by adding ontology formatting rules, and P3’s chain-of-thought instructions add a smaller gain.

Table 2. Prompt ablation Stage 1 (GPT-5-mini, 5 examples per framework, controlled). Macro- F_1 averages class-, object-property-, and data-property-level F_1 across all four frameworks.

Prompt Description		Cls F_1	Obj F_1	Data F_1	Macro F_1
P0	Baseline (schema only)	0.913	0.915	0.958	0.929
P1	– detailed instructions	0.919	0.890	0.893	0.901
P2	+ explicit ontology rules	0.936	0.949	0.985	0.957
P3	+ chain-of-thought reasoning	0.950	0.921	0.948	0.940
P4	+ self-validation checklist	0.967	0.963	0.948	0.959
P5	+ framework-specific terms	0.956	0.967	0.930	0.951

In Stage 2, P2, P4, and P5 are each run over the full per-framework example set (Table 3). On this larger sample, P5 outperforms P2 and P4 (macro- F_1 0.954 versus 0.881 and 0.935), establishing P5 as the best-performing prompt overall and the prompt selected for the model comparison. This reversal between stages is consistent with P5’s framework-specific rules providing greater marginal benefit as source-code diversity grows, while P4’s generic validation checklist already suffices on small, homogeneous samples.

Table 3. Prompt ablation Stage 2 (GPT-5-mini, 6–20 examples per framework, full sample).

Prompt Description		Cls F_1	Obj F_1	Data F_1	Macro F_1
P2	+ explicit ontology rules	0.867	0.868	0.908	0.881
P4	+ self-validation checklist	0.940	0.950	0.915	0.935
P5	+ framework-specific terms	0.952	0.965	0.945	0.954

Model Comparison (EQ3) Table 4 compares the three backends on CrewAI with prompt P5 (16 examples each, all valid). Under the best prompt, all three converge to high quality: the macro- F_1 spread is only 0.013 points (0.974–0.987). GPT-5-mini leads or ties on every dimension, with perfect data-property F_1 and the highest object-property F_1 (0.967); Claude Opus 4.8 matches its data-property score and trails overall by 0.007; Gemini 2.5 Flash trails by 0.013 but is fastest at 26.3 s per file, under half of GPT-5-mini’s 54.0 s, making it the most cost-efficient choice when latency matters. Object-property F_1 (0.950–0.967) remains the weakest, most discriminating dimension, consistent with the relation-extraction bottleneck above.

Table 4. Model comparison on CrewAI with prompt P5 (16 examples each, all valid). Avg. time is mean wall-clock seconds per generated file.

Model	Cls F_1	Obj F_1	Data F_1	Macro F_1	Avg. time (s)
GPT-5-mini	0.994	0.967	1.000	0.987	54.0
Claude Opus 4.8	0.990	0.950	1.000	0.980	35.1
Gemini 2.5 Flash	0.984	0.950	0.990	0.974	26.3

Semantic Conformance Six of the 50 P5/GPT-5-mini files were excluded (no genuine agentic implementation) and three failed parsing, leaving 41 evaluable files. Overall, 30 of 41 (73.2%) are fully conformant; LangGraph is fully conformant (5/5), while the highest non-conformance rates are AutoGen (3/6) and Mastra AI (5/15). Non-conformant files fail at most 2 of 49 rules; 71.9% of all violations stem from `:requiresResource` and `:producedResource` pointing at a `:Tool` instead of the required `beam:Resource`. The higher rates for AutoGen and Mastra AI reflect their richer resource usage patterns: AutoGen examples frequently pass data objects between agents as resources, and Mastra AI workflows explicitly define tool outputs as inputs to subsequent steps – both idioms prompt the model to use `:requiresResource/producedResource`, where it then assigns the already-declared `:Tool` individual rather than a separate `beam:Resource`. LangGraph’s full conformance corresponds to its examples’ simpler, graph-edge-driven data flow, which does not trigger these properties. In practical terms, SPARQL queries over agents, tasks, goals, and workflow structure remain unaffected; only resource-flow queries encounter incorrect range values in non-conformant files. Tightening the prompt to explicitly distinguish a consumed/produced resource from a tool an agent uses would eliminate this violation class. All six warnings are missing `:hasWorkflowStep` links on `:WorkflowPattern` individuals.

Listing 1.2 shows the SHACL shape underlying the dominant violation class together with an illustrative violating instance from the CrewAI *Game Builder Crew* pattern, consistent with the one violation reported for that file in the

Table 5. SHACL conformance summary by framework (P5/GPT-5-mini corpus: 44 files remaining after exclusions, of which 41 are evaluable).

Framework	Files	Conformant	Non-conformant	Parse error
AutoGen	6	3	3	0
CrewAI	16	12	3	1
LangGraph	5	5	0	0
Mastra AI	17	10	5	2
Total	44	30	11	3

project’s SHACL evaluation reports: `:requiresResource` points at a `:Tool` individual where a `beam:Resource` is required.

```

1 :TaskShape a sh:NodeShape ;
2   sh:targetClass :Task ;
3   sh:property [ sh:path :requiresResource ; sh:class beam:Resource ;
4     sh:severity sh:Violation ; ] ;
5   sh:property [ sh:path :producedResource ; sh:class beam:Resource ;
6     sh:severity sh:Violation ; ] .
7
8 :code_review_task a :Task ;
9   :performedByAgent :qa_engineer_agent ;
10  :requiresResource :static_analysis_tool .
11  # Violation: typed :Tool, not beam:Resource
12
13 :static_analysis_tool a :Tool ;
14   :hasCapability :capability_lint_python_code .

```

Listing 1.2. SHACL shape enforcing that `:requiresResource`/`:producedResource` must reference a `beam:Resource` individual (top), and an illustrative violation where `:static_analysis_tool` is typed `:Tool` instead (bottom).

Taken together with the completeness results above, these findings indicate that the extraction pipeline’s residual error concentrates less in missing the right ontology terms (already high per the macro- F_1 scores) and more in a narrow, systematic misuse of two specific relations. In practical terms, the 26.8% of files with at least one violation still pass all other shape constraints; a downstream SPARQL query over `:requiresResource` or `:producedResource` would encounter incorrect range values in those files, but queries over agents, tasks, goals, tools, and workflow structure – the majority of the graph – would be unaffected. The prompt guidance for `:requiresResource` / `:producedResource` can be tightened to explicitly distinguish “a `beam:Resource` consumed or produced by a task” from “a `:Tool` the agent uses,” which would eliminate this violation class independently of any change to completeness-oriented prompting.

6 Discussion

Ontology-grounded prompt engineering is the dominant lever on extraction quality Each added layer of ontology grounding, structured guidance, and validation instructions raises macro- F_1 (Table 2), and framework-specific mapping guidance

pushes it highest once source-code diversity grows (Table 3). The advantage of that guidance narrows on small, homogeneous samples, where a generic validation checklist already suffices – consistent with framework-specific rules paying off more as the corpus grows more heterogeneous.

Relation extraction is the principal bottleneck, not entity extraction Object-property F_1 is consistently the lowest and most variable metric. Relations stated directly in source code (`:performedByAgent`, `:useLanguageModel`) are recovered reliably, whereas relations that require connecting information spread across several entities (`:hasTeamGoal`, `:hasAgentConfig`, `:nextStep`) are the primary residual gap: the graphs tend to contain the right entities but miss some connections between them. The SHACL results reinforce this, showing the dominant violation is not a missing relation but a *mistyped* one (`:requiresResource` pointing at a `:Tool` instead of a `:Resource`).

Model choice is a secondary lever behind prompt design Under the best prompt, the three evaluated models converge closely (Table 4), with only specialized differences remaining – GPT-5-mini on object-property extraction and Gemini 2.5 Flash on throughput. This matches the prompt ablation’s larger effect size: prompt choice shifts macro- F_1 by 0.073 across the Stage-2 variants (Table 3), against only 0.013 across models on the same CrewAI data (Table 4) – roughly a five-fold difference in leverage. The comparison is currently scoped to CrewAI; extending it to LangGraph, AutoGen, and Mastra AI would test whether this ranking holds across frameworks with different source-code structure and complexity.

Schema anchoring keeps hallucination near zero Hallucination is a well-documented risk in LLM-generated text [19]; anchoring generation to the fixed AgentO schema confines output to a closed vocabulary despite this risk. Property hallucination is absent outright: across all 265 generated files, zero object- or data-property false positives are recorded. Class hallucinations are rare (22 of 265 files) and concentrate overwhelmingly on a single plausible-but-undefined concept, `:Context` (21 of the 22 cases). The predicate vocabulary is well-specified, while the class hierarchy is sparse enough to occasionally attract a borderline concept.

Syntax failures are a stochastic artefact, not a framework weakness The 4.2% of files that fail to parse share a single root cause: a single-quoted rather than triple-quoted delimiter around a long, embedded multi-line string, distributed across models, frameworks, and prompts – pointing to a targeted formatting fix rather than a fundamental reliability limit of LLM-guided extraction.

Recurring modelling decisions are systematic, not random Models make consistent inference choices where source information is implicit: a default language model is inferred when the source names only an environment variable, a single `:LanguageModel` individual is reused across agents sharing a configuration, and

step order is inferred from task-list position, graph edges, or call sequence rather than runtime scheduling. Missing concepts are precisely those left implicit in the source (e.g., goal text embedded in a task description drives the frequently absent `:Goal` class), while constructs with no ontology counterpart – conditional branching, lambda-based termination, speaker-selection policies – are uniformly flattened into descriptive `:Config` strings, a boundary we return to next.

7 Threats to Validity

Construct validity Precision, recall, and F_1 measure whether expected ontology terms are *present*, not whether they are used correctly; the SHACL results show this matters, since residual error concentrates in a mistyped relation invisible to completeness metrics alone. The gold standard is also defined at the framework level rather than per example, so individual files may legitimately omit concepts expected for their framework (e.g., a single-agent example has no `:Team`); per-example overrides partially address this, but manual annotation of every file would be more reliable.

Internal validity The 50 source patterns were collected from public framework example repositories rather than production systems. The model comparison (Section 5.2) uses only CrewAI, so whether the observed ranking generalizes to the other three frameworks remains to be verified.

Sampling and selection effects The 50-pattern corpus is imbalanced across frameworks (20 Mastra AI and 16 CrewAI against 8 LangGraph and 6 AutoGen), so any statistic pooled across frameworks is dominated by the two largest groups; the per-framework breakdowns reported alongside pooled results (Section 5.2 and Table 5) partially offset this, but pooled macro- F_1 and conformance figures should not be read as framework-representative on their own. Separately, the six files excluded from SHACL evaluation were identified only after inspecting the generated knowledge graphs, not chosen in advance – a post-hoc, selection-after-observation decision. The exclusion criterion (absence of any genuine agentic implementation in the source pattern) is a data-quality check independent of extraction accuracy, but the reported 73.2% conformance rate is conditional on this filtering rather than computed over the full 50-pattern corpus.

External validity Prompts are currently zero-shot; one worked example per framework could improve object-property recall by demonstrating multi-hop relation chains. The pipeline concatenates all source files without compression, so large repositories may exceed the context window and be silently truncated, and LLMs use long contexts unevenly [20]. Retrieval-augmented or chunked ingestion [29] would help at that scale and also reduce exposure to the string-escaping failure mode.

Ontology coverage of dynamic constructs Conditional branching, termination policies, and multi-turn conversation sessions have no direct ontology representation and are instead stored as opaque `:Config` strings. This is a schema gap rather than an extraction failure – the LLM correctly identifies these constructs but has no vocabulary to express them structurally.

8 Conclusion and Future Work

This paper examined whether existing, framework-specific agentic AI implementations can be automatically lifted into ontology-compliant knowledge graphs, and how reliable such a construction process is. We presented an LLM-guided pipeline that concatenates a pattern’s source files, applies an ontology-grounded prompt treating AgentO as a fixed schema and validates the resulting Turtle output. Across 50 patterns and four frameworks, the pipeline is reliable (EQ1), prompt design is by far the dominant lever on extraction quality (EQ2), model choice is secondary (EQ3), and semantic conformance is high with residual non-conformance concentrated in a single mistyped relation pair rather than broad structural breakdown (Section 5).

Future work follows directly from these findings. For the dominant remaining gap (`:requiresResource` versus tool usage), this means disambiguating prompt guidance and adding few-shot exemplars. For the recurring syntax-failure mode, stricter output formatting should eliminate it directly. The model comparison should also be extended to all four frameworks. Finally, closing the dynamic-construct coverage gap calls for concrete ontology extensions – a conditional-branching property, a termination-policy class, and a `:ConversationSession` class – that would let future extractions capture these constructs structurally.

Acknowledgments. This work was supported by the Department of Computer Science and Electronics, Universitas Gadjah Mada under the Publication Funding Year 2026. This research was funded in part by the Austrian Science Fund (FWF) 10.55776/COE12 as part of the FWF Cluster of Excellence ‘Bilateral Artificial Intelligence’ (BILAI).

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT for grammar- and spell-checking. The authors reviewed and edited the content and take full responsibility for the publication’s content.

References

1. E. Miehl, K. N. Ramamurthy, K. R. Varshney, M. Riemer, D. Bouneffouf, J. T. Richards, A. Dhurandhar, E. M. Daly, M. Hind, P. Sattigeri, et al., Agentic AI needs a systems theory, arXiv:2503.00237, 2025. <https://doi.org/10.48550/arXiv.2503.00237>.

2. N. Kshetri, Transforming cybersecurity with agentic AI to combat emerging cyber threats, *Telecommunications Policy* 49(6) (2025) 102976. <https://doi.org/10.1016/j.telpol.2025.102976>.
3. I. Okpala, A. Golgoon, A. R. Kannan, Agentic AI systems applied to tasks in financial services: Modeling and model risk management crews, *arXiv:2502.05439*, 2025. <https://doi.org/10.48550/arXiv.2502.05439>.
4. B. L. Aylak, SustAI-SCM: Intelligent supply chain process automation with agentic AI for sustainability and cost efficiency, *Sustainability* 17(6) (2025) 2453. <https://doi.org/10.3390/su17062453>.
5. N. Karunanayake, Next-generation agentic AI for transforming healthcare, *Informatics and Health* 2(2) (2025) 73–83. <https://doi.org/10.1016/j.infoh.2025.03.001>.
6. M. Gridach, J. Nanavati, K. Z. E. Abidine, L. Mendes, C. Mack, Agentic AI for scientific discovery: A survey of progress, challenges, and future directions, in: *ICLR 2025 Workshop on Towards Agentic AI for Science*. <https://openreview.net/forum?id=TyCYakX9BD>.
7. Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, et al., AutoGen: Enabling next-gen LLM applications via multi-agent conversation, in: *First Conference on Language Modeling (COLM)*, 2024. <https://openreview.net/forum?id=uAjjFFing2>.
8. CrewAI, <https://docs.crewai.com/introduction>, accessed 2025-05-11.
9. J. Wang, Z. Duan, Agent AI with LangGraph: A modular framework for enhancing machine translation using large language models, *arXiv:2412.03801*, 2024. <https://doi.org/10.48550/arXiv.2412.03801>.
10. Mastra AI, <https://mastra.ai/en/docs>, accessed 2025-05-11.
11. Regulation (EU) 2023/2854 (Data Act), 2023. <http://data.europa.eu/eli/reg/2023/2854/oj>.
12. C. Gutiérrez, J. F. Sequeda, Knowledge graphs, *Communications of the ACM* 64(3) (2021) 96–104. <https://doi.org/10.1145/3418294>.
13. S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying large language models and knowledge graphs: A roadmap, *IEEE Transactions on Knowledge and Data Engineering* 36(7) (2024) 3580–3599. <https://doi.org/10.1109/TKDE.2024.3352100>.
14. H. Babaei Giglou, J. D’Souza, S. Auer, LLMs4OL: Large language models for ontology learning, in: *The Semantic Web – ISWC 2023*, LNCS 14265, Springer, 2023, pp. 408–427. https://doi.org/10.1007/978-3-031-47240-4_22.
15. J. H. Caufield, H. Hegde, V. Emonet, N. L. Harris, M. P. Joachimiak, N. Matentzoglou, H. Kim, S. A. T. Moxon, J. T. Reese, M. A. Haendel, P. N. Robinson, C. J. Mungall, Structured prompt interrogation and recursive extraction of semantics (SPIRES): A method for populating knowledge bases using zero-shot learning, *Bioinformatics* 40(3) (2024) btae104. <https://doi.org/10.1093/bioinformatics/btae104>.
16. M. Ezzabady, F. Ieng, H. Khorashadizadeh, F. Benamara, S. Groppe, S. Sahri, Towards Generating High-Quality Knowledge Graphs by Leveraging Large Language Models, in: *Proceedings of the 29th Annual International Conference on Natural Language & Information Systems (NLDB 2024)*, Turin, Italy, 2024. https://doi.org/10.1007/978-3-031-70239-6_31.
17. H. Khorashadizadeh, N. Mihindukulasooriya, N. Ranji, M. Ezzabady, F. Ieng, J. Groppe, F. Benamara, S. Groppe, Construction and Canonicalization of Economic Knowledge Graphs with LLMs, in: *Proceedings of the International Knowledge*

- Graph and Semantic Web Conference (KGSWC), 2024. http://dx.doi.org/10.1007/978-3-031-81221-7_23.
18. A. Ekelhart, K. Kurniawan, F. J. Ekaputra, E. Kiesling, AgentO: An ontology for modeling agentic AI systems, in: *The Semantic Web – ESWC 2026*, LNCS 16550, Springer, 2026, pp. 298–320. https://doi.org/10.1007/978-3-032-25159-6_16.
 19. Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, P. Fung, Survey of hallucination in natural language generation, *ACM Computing Surveys* 55(12) (2023) 248. <https://doi.org/10.1145/3571730>.
 20. N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, Lost in the middle: How language models use long contexts, *Transactions of the Association for Computational Linguistics* 12 (2024) 157–173. <https://aclanthology.org/2024.tacl-1.9/>.
 21. T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, *Scientific American* 284(5) (2001) 34–43. <https://www.scientificamerican.com/article/the-semantic-web/>.
 22. A. Hogan, The semantic web: Two decades on, *Semantic Web* 11(1) (2020) 169–185. <https://doi.org/10.3233/SW-190387>.
 23. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing reasoning and acting in language models, in: *International Conference on Learning Representations (ICLR)*, 2023. https://openreview.net/forum?id=WE_vluYUL-X (also arXiv:2210.03629).
 24. K. Kurniawan, E. Kiesling, A. Ekelhart, CyKG-RAG: Towards knowledge-graph enhanced retrieval augmented generation for cybersecurity, in: *RAGE-KG Workshop, co-located with ISWC 2024*, CEUR-WS Vol-3950, 2024. <https://ceur-ws.org/Vol-3950/paper1.pdf>.
 25. N. F. Noy, D. L. McGuinness, et al., *Ontology development 101: A guide to creating your first ontology*, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, 2001. https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.
 26. T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, J. Zhao, PROV-O: The PROV ontology, *World Wide Web Consortium*, 2013. <https://www.w3.org/TR/prov-o/>.
 27. D. Garijo Verdejo, Y. Gil, Augmenting PROV with plans in P-Plan: Scientific processes as linked data, *CEUR Workshop Proceedings Vol-951*, 2012. <https://ceur-ws.org/Vol-951/paper6.pdf>.
 28. F. J. Ekaputra, A. Prock, E. Kiesling, Towards supporting AI system engineering with an extended Boxology notation, in: *2nd International Workshop on Knowledge Graphs for Responsible AI (KG-STAR)*, co-located with ESWC 2025, CEUR-WS Vol-4018, 2025. <https://ceur-ws.org/Vol-4018/paper3.pdf>.
 29. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks, *Advances in Neural Information Processing Systems* 33 (2020) 9459–9474. <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>.