Bachelor's thesis

Engineering: Information and Communication Technology

2025

Imad Eddine Elmouss

Structured Procedural Knowledge Extraction from Industrial Documentation Using Large Language Models

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2025 | 64 pages

Imad Eddine Elmouss

Structured Procedural Knowledge Extraction from Industrial Documentation Using Large Language Models

This thesis presents IPKE (Industrial Procedural Knowledge Extraction), a local, privacy-preserving pipeline for reconstructing procedural knowledge graphs (PKGs) from safety-critical industrial manuals.

We make two main methodological contributions:

(1) Dual Semantic Chunking (DSC), a hybrid segmentation algorithm that aligns document headings with embedding-based cohesion to reduce context fragmentation; and

(2) Two-Stage Decomposition (P3), a prompting strategy that decouples step extraction from constraint attachment.

Evaluated on marine and manufacturing SOPs, IPKE achieves 75% constraint coverage and $\Phi = 0.699$ using a quantised 7B-parameter local model, outperforming a 70B-parameter model ($\Phi = 0.439$, 50% constraint coverage) under standard prompting. This suggests that task-aligned pipeline design is more critical than model scale for extracting logic-dense industrial procedures. IPKE outputs validated, queryable PKGs intended for integration into safety-critical decision support and human-in-the-loop workflows.

Keywords:

natural language processing; information extraction; knowledge graphs; large language models; industrial documentation; procedural knowledge; semantic chunking

Content

Figures

Tables

List of abbreviations and symbols

AI	Artificial Intelligence
IE	Information Extraction
KG	Knowledge Graph
PKG	Procedural Knowledge Graph
LLM	Large Language Model
RPA	Robotic Process Automation
SOP	Standard Operating Procedure
SBERT	Sentence-BERT
ASR	Automatic Speech Recognition
CoT	Chain-of-Thought
DSC	Dual Semantic Chunking
FS-ICL	Few-Shot In-Context Learning
GGUF	GPT-Generated Unified Format
IPKE	Industrial Procedural Knowledge Extraction
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
PAGED	Procedural Graphs Extraction from Documents

1 Introduction

Industrial operations rely on extensive procedural documentation such as Standard Operating Procedures (SOPs), safety manuals, and maintenance guides. These documents encode critical workflows, but they are written for humans, not machines: task sequences, dependencies, and conditional logic remain implicit, which limits automation, digital twins, and AI-assisted decision-making central to Industry 5.0.

Prior work at Turku UAS's Cognitive Technology Lab developed an initial extraction system that identifies equipment, procedures, and safety measures from manuals. However, the output is essentially flat: it can detect "extinguisher" and "inspect monthly" but not their relationship, and it omits procedural order, conditions, and dependencies.

This thesis develops and evaluates an LLM-based pipeline for extracting procedural knowledge graphs (PKGs) from industrial documentation. The goal is to preserve (1) ordered steps, (2) conditional logic such as IF–THEN rules, and (3) constraint-to-step attachments (e.g. safety guards, preconditions, parameter thresholds), and to expose the result as a machine-actionable graph suitable for human-in-the-loop validation and downstream automation.

In practice, off-the-shelf LLM pipelines are often constraint-blind: they enumerate actions but frequently miss the rules that govern when and how those actions may be executed. This work addresses that deficiency with two main contributions:

1. **Dual Semantic Chunking (DSC):** a hybrid segmentation algorithm that aligns document headings with embedding-based cohesion, reducing context fragmentation while respecting procedural phases; and
2. **P3 Two-Stage prompting:** a schema-guided strategy that decouples step extraction from constraint attachment, reducing cognitive load on small local models and improving constraint coverage.

The resulting IPKE (Industrial Procedural Knowledge Extraction) pipeline is evaluated on three real industrial documents (marine repair, food safety, and fire-safety guidelines) using a unified metric framework centred on a Procedural Fidelity score Φ. We show that a quantised 7B-parameter local model, when paired with DSC and P3, can reach Φ≈0.7 and recover around 75% of safety constraints, outperforming a much larger 70B model under naïve prompting, while keeping all data on-premises for privacy and reproducibility.

The rest of this thesis is structured as follows. Chapter 2 reviews procedural knowledge graphs, LLM-based information extraction, and related work. Chapter 3 formalises the task and baselines. Chapter 4 presents the proposed methods (DSC, prompting strategies, graph construction, and validation). Chapter 5 reports experimental results, Chapter 6 discusses implications and limitations, and Chapter 7 concludes and outlines future directions.

2 Background & Related Work

2.1 Procedural knowledge in Industrial settings (requirements)

Industrial knowledge differs fundamentally from declarative "facts" or "definitions"; it is procedural. It dictates how to act, encoding sequences strictly bound by resources, conditions, and safety constraints. Unlike encyclopedic knowledge, industrial procedures cannot be executed blindly; actions depend heavily on context and prerequisites.

For instance, a fire-safety directive: “Inspect portable extinguishers monthly; if the pressure gauge is outside the green zone, tag and replace” contains specific logical components that a machine-actionable representation must capture:

1. sequence (Inspect → Decide → Replace),
2. guards/conditions (IF gauge IS NOT green THEN Replace),
3. equipment–action links (Extinguisher ↔ Inspect/Replace), and
4. roles/responsibilities (e.g., Operator vs Safety officer).

As Hansen et al. (2019) noted, these procedures currently remain in unstructured formats, such as PDFs or paper manuals. While human-readable, these formats are brittle for automation because the order and dependencies are implicit. Misinterpreting these dependencies can be catastrophic. The 2005 BP Texas City refinery explosion serves as a grim reminder: investigations revealed that incomplete startup procedures and deviations from protocol contributed to a disaster that killed 15 workers and injured 180 others (U.S. Chemical Safety and Hazard Investigation Board, 2007).

Figure 1 illustrates the scale of the damage caused by these procedural failures.

Figure 1. Trailers destroyed near the ISOM unit after ignition (U.S. Chemical Safety and Hazard Investigation Board n.d.).

This incident underscores the critical necessity of moving beyond static documents. To enable intelligent agents or digital twins to operate safely, we must extract these hidden workflows and rigorous dependencies into structured, unambiguous formats. The following section discusses how Information Extraction (IE) and Knowledge Graph (KG) techniques provide the technical framework to achieve this.

2.2 Information extraction and knowledge graphs

2.2.1 Definitions & scope

Information extraction (IE) transforms unstructured text into structured data. Knowledge graphs (KGs) organise entities and relations into semantic, queryable graphs. Procedural knowledge graphs (PKGs) are a specialised form of KG designed for procedural information: they explicitly represent sequential steps, conditional logic, and resource constraints, making them ideal for industrial workflows (Celino et al., 2024).

2.2.2 Why graphs for procedural knowledge

PKGs serve as the **semantic layer** between raw text and automated AI reasoning. By linking entities and relations in a structured graph, we enable AI systems to **query, trace,** and **infer** procedural logic. For example, verifying that all preconditions are met before allowing the start operation, or identifying potential conflicts between steps. This, and the relational, contextual nature of PKGs, make them a natural foundation for digital twins and intelligent agents that must interpret and execute procedures safely.

2.2.3 Representing procedural knowledge as PKGs

A minimal schema example for a representation of procedural knowledge as a knowledge graph sufficient for many industrial procedures is:

Nodes: Step, Action, Equipment/Asset, Hazard, Role, Condition, Parameter/Threshold.

Edges: Precedes(Step→Step), Requires(Step→{Equipment|Condition|Parameter}), Guards(Condition→Step), ActsOn(Action→Equipment), PerformedBy(Action→Role).

Example mapping. “Before starting pump P-101: (1) Verify valve V-23 is closed; (2) Check oil level > 80%; (3) Confirm no personnel in hazard zone; (4) Press Start.”

- Precedes: Step1 → Step2 → Step3 → Step4
- Requires: Step2 → Parameter (oil_level > 80%), Step3 → Condition (no_personnel_in_zone)
- Guards: Condition (no_personnel_in_zone) → Step4
- ActsOn: Action (VerifyClosed) → Equipment (V-23); Action (PressStart) → Equipment (P-101)

The mapping above explains why a graph schema is better than flat lists. By dividing Steps, Actions, Equipment/Assets, Conditions/Parameters, and Roles, the representation becomes executable: an agent can first check “Guards” before executing a step, then verify “Requires” resources/thresholds, then enforce “Precedes” to maintain order, and finally trace “ActsOn/PerformedBy” to assign responsibility. This also allows compliance checks (e.g., “is oil level > 80% before Start?”) and simulations (e.g., “what-if Step 2 fails?”) because conditions and dependencies are clear rather than hidden in prose.

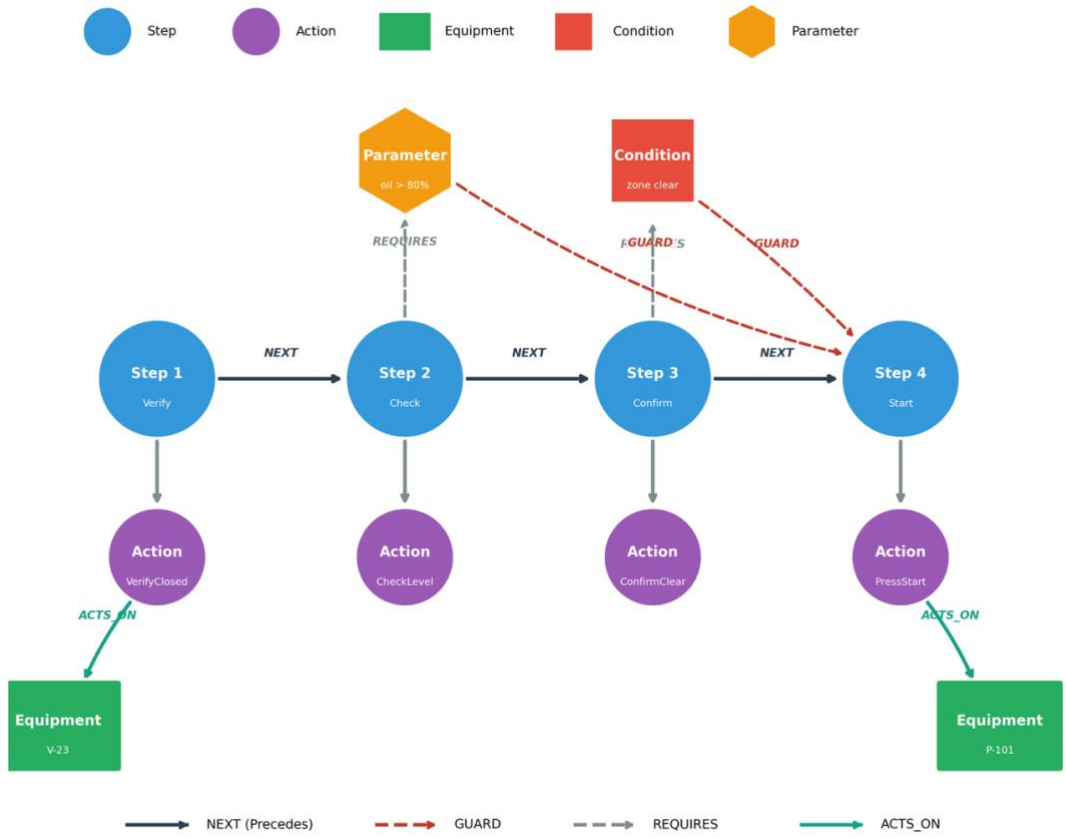


Figure 2. Visualisation of the example schema into a procedural knowledge graph.

Figure 2 illustrates this schema as a directed graph, where step nodes (blue) are connected by NEXT edges (solid arrows), and constraint nodes (red) attach to steps via GUARD edges (dashed arrows). This representation enables automated verification: before executing 'Press Start,' an agent can query all GUARD edges to ensure `oil_level > 80%` and `no_personnel_in_zone` conditions are satisfied.

2.2.4 LLM-based Information Extraction

Techniques in information extraction (IE) have evolved from manual data entry to autonomous AI-driven document understanding. In the 1990s, the focus was primarily on manual or rule-based optical character recognition (OCR), where human users corrected outputs and created templates. The advent of cloud OCR services like Tesseract simplified digitisation, yet they still relied on pattern matching without an accurate semantic understanding. Robotic Process Automation (RPA) has progressed from assisted to fully autonomous systems, integrating machine learning, computer

vision, and cognitive automation abilities to manage unstructured inputs and facilitate decision-making.

The transition from OCR/templates to RPA and then to Large Language Models (LLMs) indicates a decrease in the necessity for human scripting and an increase in semantic generalisation across various document types (Baviskar et al., 2021). Today, LLMs can generalise across documents, reason based on context rather than fixed templates, and produce structured data, such as knowledge graphs, directly from text.

In technical domains, LLMs can be trained or prompted to generate structured JSON formats that encapsulate entities and relationships, enabling the direct construction of graphs from text instead of relying on post-processing methods. Recent initiatives have fine-tuned GPT/Llama models utilising task schemata to yield document relation structures. This approach adopts a “human-in-the-loop” methodology to accelerate dataset creation and enhance accuracy. The authors also address the considerations of local self-hosting versus cloud APIs with respect to privacy, reproducibility, and cost factors that are critical for industrial applications where on-premises solutions are often favoured. Related work includes contributions from Celino, Carriero, Rula & D’Souza, and Du/PAGED.

2.3 Related Work

The evolution of automated knowledge extraction has shifted significantly from traditional supervised pipelines to generative, schema-guided approaches powered by Large Language Models (LLMs). This section reviews the convergence of Procedural Knowledge Graph (PKG) construction, LLM-based technical extraction, and the evaluation frameworks used to validate them.

2.3.1 Schema-Guided Extraction and PKG Construction

Recent surveys highlight a trend toward schema-guided, structured outputs (e.g., JSON) directly from LLMs, moving beyond simple triplet extraction (Xu et al., 2024). In technical and scientific contexts, this often involves combining LLMs with symbolic structures to produce "machine-actionable" data. For instance, Zhang et al. (2024) demonstrate that schema-aware frameworks that utilise extraction, definition, and canonicalisation can significantly reduce semantic drift and improve relation normalisation.

When applied specifically to Procedural Knowledge Graphs (PKGs), these generative capabilities offer new ways to handle unformatted text. Carriero et al. propose a prompt-based pipeline for generating PKGs, noting that the subjectivity of step identification means there is rarely a single "gold truth." To mitigate this ambiguity, Rula & D'Souza demonstrate that in-context learning (ICL) enriched with ontological definitions and few-shot examples improves procedure extraction across domains. However, industry-oriented approaches emphasise that automation alone is insufficient. Celino et al. introduce a PKO-based approach that necessitates human verification tools to address multi-modal gaps and safety-critical inaccuracies, while recent pipelines in materials science emphasise the practicality of converting semi-structured tables to KGs using validation layers and graph database integrations (e.g., Neo4j).

2.3.2 Benchmarking and Resource Constraints

As generative extraction matures, the focus has expanded to large-scale benchmarking and deployment constraints. Du et al. (2024) introduced the PAGED benchmark, providing a massive dataset of document–graph pairs to evaluate procedural alignment and graph-centric supervision at scale. While PAGED demonstrates the community's shift toward large-scale graph supervision, industrial reality often demands operations in resource-constrained environments.

Addressing this "local" gap, the LLM-TKIE framework combines OCR with prompting to emit structured data using quantised open-source models. This highlights the viability of on-premises deployments where data privacy or hardware limitations preclude the use of massive proprietary models. This distinction is critical: while benchmarks like PAGED focus on high-resource, broad-coverage extraction, practical engineering applications often require pipelines that balance model size (quantisation) with high specific accuracy.

2.3.3 Evaluation Methodologies

To assess these systems, the field has coalesced around four complementary metric families:

1- **Textual Similarity:** Metrics such as ROUGE measure lexical overlap and semantic recall relative to reference descriptions.

2- **Order Sensitivity:** Crucial for procedures, these metrics assess how well the extracted sequence preserves step adjacency and rank correlation.
3- **Graph Fidelity:** Structural metrics (e.g., Smatch or edge-type F1) evaluate the alignment of predicted entities and relations against a gold topology.
4- **Human Evaluation:** As emphasised by Carriero et al. and Celino et al., expert judgment remains essential for determining perceived usefulness, correctness, and trust.

Recent work suggests that a robust evaluation must triangulate these perspectives, combining calculating structural precision with human assessment of "usefulness" (Du et al., 2024).

2.3.4 Summary and Positioning

The existing literature mainly focuses on two extremes: extensive, broad-coverage benchmarks like PAGED, or general-purpose extraction using resource-intensive LLMs. In contrast, our work emphasises maximising procedural fidelity within the limits of small, local models. By utilising effective prompting strategies supported by a strict Knowledge Graph backend, we aim to demonstrate that high-quality, schema-compliant procedural extraction is achievable without relying on excessively large models, thereby bridging the gap between academic benchmarks and practical on-premises industrial deployment.

3 Problem Definition & Baseline Evaluation

3.1 Task formalization

Let $d \in D$ denote an industrial document (e.g., SOP, guideline, manual). Our task is to extract a machine-actionable representation of the procedures in d that document that preserves (1) the set of steps, (2) their execution order, and (3) any procedural constraints (guards/conditions and resource requirements).

Formally, we input raw text from a format-aware processing pipeline that extracts text from multiple modalities (e.g., text/PDF/Word/HTML/CSV/PPTX; OCR for images/scans; ASR for audio). In the baseline, d is processed in non-overlapping fixed-sized chunks. Furthermore, an instruction-tuned LLM processes each chunk independently and produces local chunk-level extractions that are merged into a single prediction for d.

As a presentation of the baseline extractions target, a document's procedural content is modelled by:

- A finite set of **steps** $S = \{s_1, \dots, s_n\}$ with a (partial or total) **execution order** $\pi: S \to \{1, \dots, n\}$;
- And a set of **constraints** $C = \{c_1, \dots, c_m\}$ (e.g., guards, safety, pre/post-conditions, parameter or resource requirements), with an attachment relation $attach: C \to 2^S$ linking each constraint to the step(s) it constrains.

Given $\pi(s)$ representing the execution order of step s, we define the **adjacency** set as:

$$A = \{(s_i, s_j) \in S \times S \mid \pi(s_j) = \pi(s_i) + 1\}$$

This captures all consecutive step pairs in the extracted order.

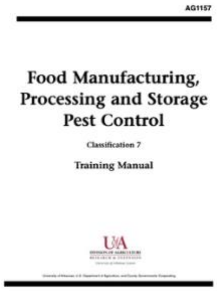
Furthermore, the system outputs primary baseline tier-A extractions, returned as a flat JSON object with:

- $steps = [\{id, text, order\}, \dots]$ where order encodes π;
- optional $constraints = [\{id, text, \dots\}, \dots]$ as free-text statements, ideally referencing step IDs, attachments are included only if the LLM produced explicit step references;
- optional $entities$ (facts/parameters/tools) used for analysis but not scored in the baseline headline metric.

3.2 Dataset & documents

We experimented on three industrial documents with **manually** curated **gold** (step lists with IDs and constraint phrases; constraint->step links when definitive). These documents are as follows:

Table 1. Document title, description and preview of our 3 dataset files

3M_OEM_SOP	A 3M Marine Standard Operating Procedure handbook covering surface preparation and application procedures (stepwise instructions)	
DOA_Food_Proc	A pest-control training manual for food manufacturing/processing/storage (regulatory + procedural content)	
op_firesafety_guideline	Fire-safety guidelines for working machines (risks, prevention, operating procedures)	

3.3 Baselines

The baseline is a local running, instruction-prompted extractor built on llama.cpp with **Mistral-7B-Instruct** (GGUF). This open-weight model strikes a balance between strong

reasoning and efficiency, making it ideal for understanding procedural text. Its compact 7B size allows us to run local reproducible experiments. Decoding uses low-temperature settings:

Table 2. Baseline model parameters (Mistral-7B-Instruct).

Setting	*Value*	*Description*
temperature	0.1	Low value chosen to minimize creativity and maximize stability; ensures deterministic and factual JSON generation.
top_p	0.9	Nucleus sampling that considers the top 90% cumulative probability mass, balancing fluency with robustness against rare, erratic completions.
max_tokens	1536	Maximum number of tokens generated per response; prevents runaway generations while allowing detailed structured outputs.
repeat_penalty	1.1	Penalizes repeated n-grams to reduce loops and redundant text in long generations.
n_ctx	8192	Model context window size; supports processing long prompts and large document chunks without truncation.
chunk_size	2000 (characters)	Text segmentation size per LLM call; fits within context limits and reduces semantic bleed between chunks.
llm_max_chunks	0 (unlimited)	No upper limit on processed chunks, ensuring complete document coverage without silent truncation.
confidence_threshold	0.8	Minimum per-item confidence, filters out weak or uncertain extractions.
quality_threshold	0.7	Document-level aggregate confidence threshold, flags low-quality or noisy extractions for manual review.

The prompt we used is short and clean:

```
"[INST] You produce lightweight procedural structure from
{document_type} documents… Return a SINGLE JSON object
with: steps (S1, S2, …), constraints (C1, …, referencing
```

`steps), and entities (supporting facts). Return only the JSON object… [/INST]”`

Inspired by Xu et al. (2024)’s description of generative IE prompts, which combine **concise task framing**, explicit **output schema definition**, and **constrained decoding** for structural consistency, the prompt is designed to extract procedural knowledge with minimal linguistic noise.

The use of explicit instruction tags (`[INST] … [/INST]`) delineates model scope, improving reproducibility and ensuring deterministic JSON output suitable for automated parsing. The schema explicitly separates procedural steps, conditional constraints, and factual entities, allowing the model to represent order and dependencies in a machine-readable yet interpretable form.

After processing all chunks in parallel, we merge the results as follows:

1. Step deduplication: Steps with > 0.9 SBERT similarity are collapsed into a single step, retaining the ID from the first occurrence.
2. Constraint deduplication: Similarly, near-duplicate constraints are merged.
3. Order reconstruction: Steps are re-sorted by their original document position (character offset), and order indices are reassigned sequentially.
4. Constraint attachment: If a constraint references a step ID that was deduplicated, the reference is updated to the canonical ID.

This setup produces a flat list of steps (implicitly ordered by list position), accompanied by optional free-text constraints and contextual entities supporting each step.

3.4 Minimal schema

To support a better, improved structured extraction, we define a minimal procedural schema, where names are lowercase at the JSON level:

Node (or vertex) **types**:

1- `step` executable instructions; fields: `{id, text, order?, context?}`
2- `condition` guard/requirement/safety note; fields: `{id, expression, type?, context?}`
3- `equipment` tool/material/asset; fields: `{id, name, model?, category?, vendor?, notes?}`
4- `parameter` named parameter/threshold, fields: `{id, name, value?, unit?, range?, notes?}`

(Fields followed by a question mark indicate they are optional (e.g., `value?`))

Edge (or link) **types:**

1- `next` temporal adjacency between steps(primary)
2- `condition_on` a condition attached to a step it constrains (primary)
3- Reserved: `uses, has_parameter, requires, produces, references, alternative_to`

Example:

```
{
  "document_id": "toy_doc",
  "nodes": [
    { "id": "S1", "type": "step", "text": "Shut off main power.", "order": 1 },
    { "id": "S2", "type": "step", "text": "Open the service panel.", "order": 2 },
    { "id": "C1", "type": "condition", "text": "Wear insulated gloves." }
  ],
  "edges": [
    { "source": "S1", "target": "S2", "type": "next" },
    { "source": "C1", "target": "S1", "type": "condition_on" }
  ]
}
```

Figure 3. Example of a JSON lightweight procedural knowledge graph.

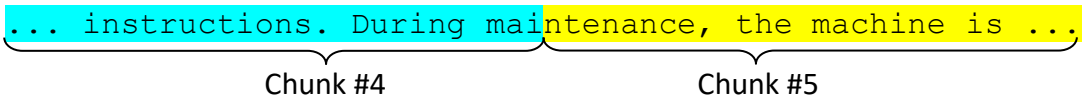
4 Proposed Methods

4.1 Document preprocessing & semantic chunking

The baseline implementation exhibits significant structural deficiencies, firstly in **chunking**; the splitting of documents into individual chunks (groups of text) to send the full text to the LLM for processing one chunk at a time. The issue lies in the chunking method itself; the current pipeline splits the document into fixed-size text chunks, defaulting to 2000 characters, or approximately 500 tokens (1 token equals 4 characters). Even though this is good in how it improves the throughput, it still raises a big problem:

It breaks cross-chunk continuity; sentences, lists, or even words can be cut mid-way, and the context established in one chunk is unavailable to the next, since they are analysed individually, separated, and presented one after the other.

For example, in the document “op_firesafety_guideline.pdf” two consecutive chunks (the fourth and fifth ones) show a sentence split across the boundary, even on the word level, “maintenance” was split into “.. mai” and “ntenance …” as the table shows:



This fragmentation leads to a loss of meaning and logic within chunks, as well as across relationships between entities across different chunks, leading us to question how we can effectively chunk large documents more efficiently without dramatically increasing chunk size or losing meaning across chunk boundaries.

4.1.1 Breakpoint-based semantic chunking

A standard method for mitigating context loss is to use **chunk overlap** in our text splitting. This is achieved by overlapping a few tokens/characters/sentences between neighbouring chunks, both of which contain the shared context. For example, if Chunk#1 ends with a sentence and Chunk#2 starts mid-idea, both chunks can include the shared sentence, so whichever chunk is returned, the relevant idea is not lost.

Example:

We could set an input to be five bullet points about an industrial/procedural text:

```
1. Turn off the main valve before starting any maintenance.
2. Remove the protective cover from the assembly.
3. Inspect the gaskets for signs of wear or leakage.
4. Replace gaskets if any damage is found.
5. Reinstall the protective cover securely.
```

Let us use two sentences for a chunk size, paired with a one-sentence chunk overlap (so the last sentence of the previous chunk also appears at the start of the next one). We then get the following results:

```
1. Turn off the main valve before starting any maintenance.
2. Remove the protective cover from the assembly.
3. Inspect the gaskets for signs of wear or leakage.
4. Replace gaskets if any damage is found.
5. Reinstall the protective cover securely.
```

Compared to, with the overlap (1 sentence):

```
1. Turn off the main valve before starting any maintenance.
2. Remove the protective cover from the assembly.
3. Inspect the gaskets for signs of wear or leakage.
4. Replace gaskets if any damage is found.
5. Reinstall the protective cover securely.
```

The dark blue highlighted zones are in common with both the previous and next chunks; this is especially beneficial in the context of industrial or procedural text, where each step depends heavily on the previous step's context. In this example, for instance, our system gets both "Remove the protective cover from the assembly." AND "Inspect the gaskets for signs of wear or leakage," so we know what step we are on and what to do next.

Chunk overlap prevents boundary context loss but can not capture deeper non-adjacent sentence relationships, as it's a static splitting method. This leads to embedding-based chunking, which groups related content by meaning rather than proximity or item count.

Embeddings encode semantic meaning; comparing them helps infer relationships and identify text clusters. Our study finds split points between sentences: starting from sentence #1, compare its embedding to subsequent sentences for significant differences. When the embedding distance exceeds a threshold, it marks a new semantic section. This method, used in frameworks like LangChain and LlamaIndex, maintains semantic coherence in retrieval-augmented systems. We can formalise this as an optimisation problem, following Alemi and Ginsparg's framework for text segmentation using semantic embeddings. Consider a document of n sentences, with

$e_k \in \mathbb{R}^d$ representing the semantic content of the sentence k in a d-dimensional embedding space (Alemi & Ginsparg, 2015).

For any segment of the document spanning from sentence index u to v (inclusive, where $1 \leq u < v \leq n$), we define a **cohesion score:**

$$c(u, v) = \frac{1}{v-u} \sum_{k=u}^{v-1} sim(e_k, e_{k+1})$$

Where $sim(e_k, e_{k+1})$ measures the cosine similarity between the embeddings of adjacent sentences k and $k+1$. This score quantifies the average semantic continuity within the segment; a high value indicates the sentences are semantically tightly coupled, while a lower value suggests a topic shift.

Our goal is to split the document sequence $[1, n]$ into a set of disjoint segments $S = \{s_1, s_2, \ldots, s_m\}$ that partitions the text while maximising the following objective function:

$$J(S) = \sum_{s \in S} c(s) - \lambda \mid S \mid$$

Here, $\mid S \mid$ denotes the number of segments (chunks), and $\lambda > 0$ is a regularisation parameter (break penalty). The objective rewards high internal cohesion within segments ($c(s)$) while penalising excessive fragmentation via the cost λ for each resulting chunk. This formulation is inspired by standard text segmentation literature (e.g., Alemi & Ginsparg, 2015), where λ controls the granularity: small values favour shorter, more numerous chunks, while larger values encourage fewer, longer chunks.

This optimisation problem can be solved efficiently using dynamic programming. We define $DP[j]$ as the maximum objective value achievable for segmenting the document prefix spanning sentences 1 to j. The recurrence relation is defined as:

$$DP[j] = max_{0 \leq i < j}\{DP[i] + c(i+1, j) - \lambda\}$$

With the base case:

$$DP[0] = 0$$

In this recursion, i represents the split point (the last sentence of the previous optimal segmentation). Consequently, the term $c(i+1, j)$ represents the cohesion score of the current candidate segment spanning from $i+1$ to j. The algorithm iterates through all valid start positions i to find the optimal boundary that maximises the cumulative score up to j.

4.1.2 Dual Semantic Chunker (DSC)

We define the chunking process as an optimisation problem following the text segmentation principles established by Alemi and Ginsparg (2015). Consider a document D as a sequence of sentences $X = \{x_1, x_2, \dots, x_n\}$. Our goal is to find a segmentation $B = \{b_1, b_2, \dots, b_k\}$ that maximises semantic similarity within each chunk while maintaining the document's hierarchical structure.

We define the **Cohesion score** $H(b)$ for a block b containing sentences from index i to j as the average cosine similarity between adjacent sentence embeddings, utilising the SBERT framework (Reimers & Gurevych, 2019):

$$H(b_{i:j}) = \frac{1}{j-i} \sum_{k=i}^{j-1} \cos\,(e_k, e_{k+1})$$

Where e_k is the vector embedding of sentence x_k. The DSC strategy maximises the global objective function $J(B)$:

$$J(B) = \sum_{b \in B} H(b) - \lambda \mid B \mid$$

Where λ is a hyperparameter penalising excessive fragmentation, a higher λ is preferred in industrial contexts to prevent the fragmentation of complex safety instructions into non-functional micro-chunks. Additionally, DSC enforces a hierarchical rule stating that a split point p is valid only if it aligns with a semantic breakpoint or a structural header (e.g., "`### Safety Precautions:`"). This hybrid objective ensures that the resulting chunks are both semantically coherent and aligned with the document's logical structure, addressing the "Lost in the Middle" phenomenon commonly observed in long-context retrieval (Liu et al., 2024).

Hierarchical Constraint: In practice, we modify the DP search to enforce that segment boundaries align with heading positions. Specifically, if position j is immediately after a heading (e.g., '`### Safety Precautions:`'), we add a bonus term β > 0 to DP[j]:

$$DP[j] = max_{\{i<j\}}\{\, DP[i] + c(i,j) - \lambda + \beta \cdot \mathbb{1}\}$$

where $\mathbb{1}[\cdot]$ is an indicator function and j is header position. This encourages splits at structural boundaries even when semantic coherence is locally high. We set β = 0.2 empirically, balancing semantic and structural alignment.

Putting it all together, the two stages are illustrated in Figure 4:

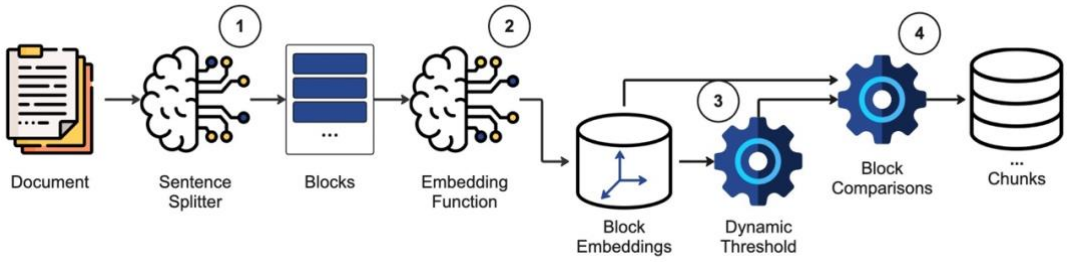


Figure 4. Overview of the Dual Semantic Chunker.

4.2 Prompting & Schema-Constrained JSON Emission

4.2.1 Prompting baseline

Having addressed how the documents are divided into semantically stable chunks with good information retention, the following methodological challenge is how to instruct the model. The limitation is no longer *what* the model reads, but *how* it is instructed to read. This makes prompt design our primary method for guiding the LLM's behaviour, to enforce structure and minimise variability in our procedural knowledge extraction.

Following the application of Dual Semantic Chunking (DSC), our LLM's calls should now receive **coherent**, **self-contained textual segments** that capture a locally consistent part of the text. Our baseline system currently runs on a single-shot, instruction-following prompt that:

- Gives the model a **role** ("`you produce lightweight procedural structure`")
- Specifies a **strict JSON schema** (steps/constraints/entities)
- Enforces **ID rules** (S1, S2, …; C1, C2, …)
- Includes **guidelines** (steps in execution order, constraints as requirements, concise phrasing)
- Forces **JSON-only** output,
- It is used in **parallel** per chunk, then merged (meaning independent simultaneous chunk processing and extraction, and only merging all the results after all chunks complete)

To better illustrate this workflow, we can take this example to showcase what the model receives as input. Let us say we were dealing with a fire safety document, and one chunk contains the following 2000 chars:

"**Employees must** inspect fire extinguishers **monthly**, checking the pressure gauge to ensure it is within the

green zone of **12** to **18 bar** (**174** to **261 psi**). If the gauge shows **low pressure, replace** or **recharge** the **extinguisher immediately.** All personnel **should be trained** on the **proper use** of **extinguishers before operating** them. The **PASS technique** should be **followed: Pull** the **pin, Aim** at the **base of the fire, Squeeze** the **handle,** and **Sweep side to side. Extinguishers must** be **mounted** at **accessible heights,** with **no more than 5 feet between** the **floor** and the **top of the extinguisher.**"

The exact prompt sent to the LLM, in our baseline system, would be:

```
[INST] You produce lightweight procedural structure from manual documents.

Read the provided text and return a SINGLE JSON object with concise fields:
{
  "steps": [
    {"id": "S1", "text": "Action statement written as an imperative.", "type": "procedure_step",
"confidence": 0.9},
    {"id": "S2", "text": "Next ordered action.", "type": "procedure_step", "confidence": 0.88}
  ],
  "constraints": [
    {"id": "C1", "text": "Condition, warning, or requirement.", "steps": ["S1"], "confidence":
0.85}
  ],
  "entities": [
    {"content": "Supporting fact or measurement.", "type": "specification", "category":
"distance_requirement", "confidence": 0.9}
  ]
}

Guidance:
- Keep IDs sequential (S1, S2, ..., C1, C2, ...).
- Steps must be actual actions or instructions in execution order.
- Constraints capture requirements, cautions, or prerequisites. Refer to step IDs when possible.
- Include additional granular facts in `entities` when helpful (measurements, tools, materials).
- Prefer short phrases; avoid restating large paragraphs.

Text:
"""Fire extinguishers must be inspected monthly. Check the pressure gauge to ensure it is in the
green zone. If the gauge shows low pressure, replace or recharge the extinguisher immediately.
All personnel should be trained on proper extinguisher use before operating them. The PASS
technique should be followed: Pull the pin, Aim at the base of the fire, Squeeze the handle, and
Sweep side to side. Extinguishers must be mounted at accessible heights, no more than 5 feet from
the floor to the top of the extinguisher."""

Return only the JSON object described above with no commentary.
[/INST]
```

The model responds with this output under JSON format:

```
{
  "document_id": "cc1c61e0278f5d58",
  "document_type": "text",
  "entities": [
    {
      "content": "Fire extinguisher pressure within green zone: 12-18 bar (174-261 psi).",
      "entity_type": "specification",
      "category": "pressure_requirement",
      "confidence": 0.95,
      "context": "Employees must inspect fire extinguishers monthly, checking the pressure gauge
to ensure it is within the green zone of 12 to ..."
    }
  ],
  "steps": [
    {
      "id": "S1",
      "text": "Check pressure gauge of fire extinguisher monthly.",
      "order": 1,
      "confidence": 0.95
    },
    {
      "id": "S2",
      "text": "Ensure pressure is within green zone (12-18 bar).",
      "order": 2,
      "confidence": 0.9
    },

    ... // 6 more steps

    {
      "id": "S7",
      "text": "Keep minimum 5 feet between floor and top of extinguisher.",
      "order": 7,
      "confidence": 0.9
    }
  ],
  "constraints": [],
  "confidence_score": 0.95,
  "processing_time": 27.39,
  "strategy_used": "llama.cpp",
  "metadata": {
    "file_format": ".txt",
    "file_name": "fire-safety-test.txt",
    "content_length": 579,
    "entities_extracted": 1,
    "steps_extracted": 7,
    "constraints_extracted": 0
  }
}
```


Although the model made seven step extractions ("steps_extracted": 7) and identified one specific spec entity ("entities_extracted": 1), it still failed to extract any constraints, despite the text clearly containing many constraints and requirements (must, should, if…then, height limits), like:

"Employees **must** … monthly"
"If the gauge shows **low pressure, replace** or **recharge** the **extinguisher immediately**"
"All personnel **should be trained**…" etc…

To investigate the cause of this extraction failure, an error analysis reveals that the model received a **well self-contained snippet**, yet the extraction was still not ideal.

Therefore, it cannot be a chunking issue. Whether the entire paragraph is presented as one chunk or the whole document, the result remains the same.

This indicates that our current prompt under-specifies the notion of 'constraint', and that the model instead overgeneralizes it as more procedural steps. Since we used zero-shot without examples, the model never had a demonstration of what a constraint *looks like* compared to a step. This aside, another factor could be how the model does not reason in the process of deciding what is what; no Chain-of-Thoughts (CoT) means our model never explicitly thinks "this is a rule that modifies a step", for example Another possible, simpler reason is that the definition of `constraints` in our JSON schema might just be too vague or buried "`constraints capture requirements and prerequisites`" is too abstract.

This drives us to revisit not only the schema, but the entire prompting strategy.

Carriero et al. (2024) outline three key prompting types in the LLM field: zero-shot instruction, few-shot in-context learning, and chain-of-thought prompting (para. X). Each offers a different way to instruct models. Zero-shot prompting is the simplest, involving only natural language instructions and possibly a task description, with no examples, like: "I `need you to ...`" This approach is common among users and our baseline system, which uses a role description ("`You produce lightweight procedural structure from manual documents`") and a JSON schema ("`Read the provided text and return a SINGLE JSON object with concise fields:`"), but no demonstration of "steps" or "constraint".

By contrast, **few-shot prompting** augments the instructions by adding a small yet highly beneficial detail: a few **input-output examples** within the prompt, teaching the model how to behave on the fly. Brown et al. (2020) demonstrated that in-context learning enables LLMs to generalise to new tasks with only a handful of examples, without requiring any parameter updates.

A third option is to give the model more room to reason and reflect on its extractions. **CoT**, or **Chain-of-Thought** prompts, explicitly ask the model to produce intermediate reasoning steps before giving any final answer. Wei et al. (2022) and Kojima et al. (2022) demonstrate that generating such step-by-step rationales can substantially improve performance on multi-step reasoning tasks, both in few-shot and in zero-shot settings. While these models are mostly reported for very large models, the underlying core is appealing for our use case. Before emitting a structured JSON, the model can explicitly reason about which sentences are "actions" and which ones are "rules that modify actions", rather than placing everything under steps.

Finally, recent research on **structured outputs and schema-constrained decoding** examines methods to make LLMs consistently generate machine-readable data that

adheres to a user-provided schema, sometimes even optimising the schema itself for extraction quality. Our baseline already implements a simple schema in the prompt and performs validation after the fact. However, as the fire example shows, the semantics of `constraints` are still under-specified: "requirements and prerequisites" is too vague, and nothing in the prompt prevents the model from simply treating every modal ("must", "should") as just another procedural step.

Building on these observations, we developed a series of **prompting strategies** that gradually enhance the guidance provided to the model:

- **P0: Zero-shot Guided JSON Instruction Prompting (Baseline)**
  Our baseline regime, as described above, involves a single-shot, schema-guided instruction prompt with role assignment, explicit JSON fields (`steps`, `constraints`, `entities`), ID rules, and guidance on conciseness, although without any examples or explicit reasoning, and is used independently on each chunk.
- **P1: Few-shot In-Context Learning (FS-ICL)**
  P1 enhances **P0** with one or two annotated examples. Each example includes a short document fragment and its gold JSON extraction, where some obligations and guard conditions are intentionally encoded as constraints rather than steps. The rest of the schema and guidance remain the same. This approach tests whether in-context demonstrations help the model understand the difference between "do X" and "X must hold while/before doing Y," thereby improving constraint coverage and attachment.
- **P2: CoT-augmented JSON Prompting (CoT)**
  In P2, the model is first asked to reason in natural language about the procedural structure ("list the likely actions, then identify which sentences state conditions, rules, or safety requirements and which steps they modify"), and only then to produce the final JSON object. During decoding, we discard the free-text rationale and only parse the JSON block. This approach aims to encourage the model to explicitly classify sentences as actions versus constraints before committing to a structured representation, potentially enhancing ordering metrics and constraint identification.
- **P3: Two-stage, schema-aware prompting**
  Finally, **P3** breaks down the extraction into two LLM calls. In Stage 1, the model extracts only steps with stable IDs and local successor relations. In Stage 2, it receives the list of steps (IDs and texts) and is asked to produce only constraints and entities, with a clear rule that every constraint must reference at least one existing step ID. Outputs are validated against a JSON schema, and invalid references are rejected or corrected. This process enables tighter control over constraint attachment and prepares data for downstream PKG construction.

In the experimental section 5.4, we maintain the chunking strategy of our best semantic chunker (DSC) and compare P0–P3. We expect that the few-shot and two-

stage regimes (P_1 and P_3) will primarily increase constraint coverage and attachment quality, while CoT-style prompting (P_2) might yield more globally consistent step ordering and dependency structure.

4.2.2 P1 – Few-shot in context learning (FS-ICL)

Following Brown et al. (2020)'s definition, an in-context learning (ICL) task is one in which the strategy does not update the pretrained model's weights. Instead, it conditions the model on a sequence of demonstrations; typically, an input and its ground truth label, both formatted using a specific pattern and a verbalizer. Thus, FS-ICL refers to in-context learning with only a few training examples, similar to the following two examples:

Example 1:

```
Input fragment:
'If needed, mix 3M™ Marine Blister Repair Filler and apply. Fill 85% of the cavity. After 30 minutes,
feather the repair with 3M™ Hookit™ 775L Cubitron™ II abrasive disc, grade 80+.'
Gold extraction:
{
  "steps": [
    {
      "id": "S1",
      "text": "Mix 3M Marine Blister Repair Filler",
      "order": 1,
      "context": "filler repair"
    },
    {
      "id": "S2",
      "text": "Apply filler to cavity",
      "order": 2,
      "context": "filler repair"
    },
    {
      "id": "S3",
      "text": "Feather the repair with abrasive disc",
      "order": 3,
      "context": "filler repair"
    }
  ],
  "constraints": [
    {
      "id": "C1",
      "expression": "Apply only if needed",
      "type": "guard",
      "attached_to": ["S1", "S2"]
    },
    {
      "id": "C2",
      "expression": "Fill level: 85% of cavity",
      "type": "parameter",
      "attached_to": ["S2"],
      "parameters": {
        "fill_level": {"value": 85, "unit": "%", "target": "cavity_volume"}
      }
    },
    {
      "id": "C3",
      "expression": "Wait 30 minutes before feathering",
      "type": "temporal",
      "attached_to": ["S3"],
      "parameters": {
        "wait_time": {"value": 30, "unit": "minutes"}
      }
    }
  ],
  "entities": [
    {"id": "E1", "name": "3M Marine Blister Repair Filler", "category": "material"},
    {"id": "E2", "name": "3M Hookit 775L Cubitron II abrasive disc", "category": "tool"},
    {"id": "E3", "name": "grade 80+", "category": "parameter", "parameters": {"grade": 80, "operator": "+"}}
  ]
}
```

Figure 5. Marine Gelcoat Repair (FS-ICL - Technical Procedure from the “3m marine OEM SOP” doc in our dataset).

In this example, we parsed values such as "85%" and “30 minutes” into numerical fill levels and wait times with associated units, and "80+" into a structured grade with an operator. This exemplifies FS-ICL's fundamental teaching signal: the model learns that modal language, such as "If needed," corresponds to guard constraints (C1) rather than

procedural steps. At the same time, quantitative limits are converted into parameter constraints (C2, C3) with machine-readable values.

Example 2:

```
Input fragment:
'Inspect and clean the engine compartment and surroundings daily when operating in dusty or fire-hazard
conditions (e.g., timber industry, peat processing).'
Gold extraction:
{
  "steps": [
    {
      "id": "S1",
      "text": "Inspect engine compartment and surroundings",
      "order": 1,
      "context": "preventive maintenance"
    },
    {
      "id": "S2",
      "text": "Clean engine compartment and surroundings",
      "order": 2,
      "context": "preventive maintenance"
    }
  ],
  "constraints": [
    {
      "id": "C1",
      "expression": "Frequency: daily",
      "type": "schedule",
      "attached_to": ["S1", "S2"],
      "parameters": {
        "frequency": {"value": 1, "unit": "day"}
      }
    },
    {
      "id": "C2",
      "expression": "Applies only in dusty or fire-hazard environments",
      "type": "environmental_guard",
      "attached_to": ["S1", "S2"],
      "parameters": {
        "conditions": ["dusty", "fire_hazard"],
        "examples": ["timber_industry", "peat_processing"]
      }
    }
  ],
  "entities": [
    {"id": "E1", "name": "engine compartment", "category": "component"}
  ]
}
```

Figure 6. Fire Safety Inspection (FS-ICL - Fire Safety Inspection from the "OP firesafety guideline" doc in our dataset).

In this second example, the temporal adverb "daily" and the environmental condition "when operating in dusty conditions" are extracted as separate constraints (C1, C2) associated with both steps. This FS-ICL example directly addresses the baseline's inability to distinguish between schedules and environmental guards, as opposed to core actions. By normalising "daily" to a frequency parameter and structuring conditions as machine-readable tags, the model maintains procedural logic and avoids hallucinating unnecessary steps such as "Check if dusty". This separation of constraints and steps is crucial for safety-critical automation.

4.2.3 P2 – CoT

Unlike the direct-answering methods (P0 and P1), Chain-of-Thought (CoT) guides the Large Language Model (LLM) to perform intermediate reasoning steps before producing its final output. Wei et al. (2022) show this improves performance on complex tasks. CoT prompts embed reasoning steps within the prompt, enabling the model to “decompose” problems. In our pipeline, the P2 strategy updates the prompt to explicitly request a natural language analysis before generating a JSON object.

1- **CLASSIFY**: Determine if the text segment is "Descriptive" (facts, definitions) or "Procedural" (instructions). If descriptive, the model is instructed to halt extraction, significantly reducing false positives.
2- **FILTER**: Identify the actor. The model distinguishes between human actions (e.g., "Operator inspects valve") and non-human events (e.g., "Beetle eats grain"), keeping only the former.
3- **ATTACH**: Scan the context for preconditions ("before", "ensure") and safety guards ("must", "if"), explicitly linking them to the candidate steps.
4- **EMIT**: Finally, convert the reasoned analysis into the strict JSON schema.

```
[INST] You are an expert technical analyst. Your goal is to extract a structured procedural graph
from the provided industrial text.

Perform the following reasoning steps in natural language before generating the final JSON:

1. CLASSIFY: Is this text "Descriptive" (facts, biology, definitions) or "Procedural"
(instructions, rules, maintenance steps)?
   - If Descriptive, explicitly state that no steps should be extracted.
   - If Procedural, identify the HUMAN actor (e.g., Operator, Technician).

2. FILTER & EXTRACT STEPS: List candidate steps (Verbs + Objects).
   - Ignore actions performed by non-humans (e.g., "The beetle eats grain" is NOT a step).
   - Ignore general observations (e.g., "Fire is dangerous").
   - Keep only actionable instructions (e.g., "Inspect the valve").

3. ATTACH CONSTRAINTS: For each identified step, scan the immediate context for:
   - Preconditions (words like "before", "after", "ensure").
   - Safety Guards (words like "must", "never", "if").
   - Quantitative specs (temperatures, distances, equipment names).
   - Explicitly link these to the specific step ID they modify.

4. EMIT JSON: Output the final result in the strict JSON schema provided, containing ONLY the
"steps", "constraints", and "entities" arrays.

Text to process:
{chunk_text}
[/INST]
```

Figure 7. P2, CoT prompt preview.

Implementation note: We will use the prompt example from Figure 7. When implementing that, the LLM outputs a reasoning thought trace followed by the desired

JSON block. Our system will automatically delete the reasoning text and retain only the valid JSON object found between code fences.

4.2.4 P3 – Two-Stage

While CoT improves reasoning within a single context, it still requires the model to handle analysis, step extraction, ordering, and constraint attachment all at once within a limited context window, increasing the cognitive load or pressure on the model, which is especially problematic for relatively small models like our Mistral-7B with fewer parameters, being vulnerable to instruction drift where the model successfully reasons about the text but fails to adhere to the strict schema or ID-linking rules in the final output (Liu, Wang, & Zhang, 2025).

To reduce this cognitive load and attempt to avoid any attention or instruction drift, the P3 two-stage prompting strategy splits the task into two distinct sequential calls:

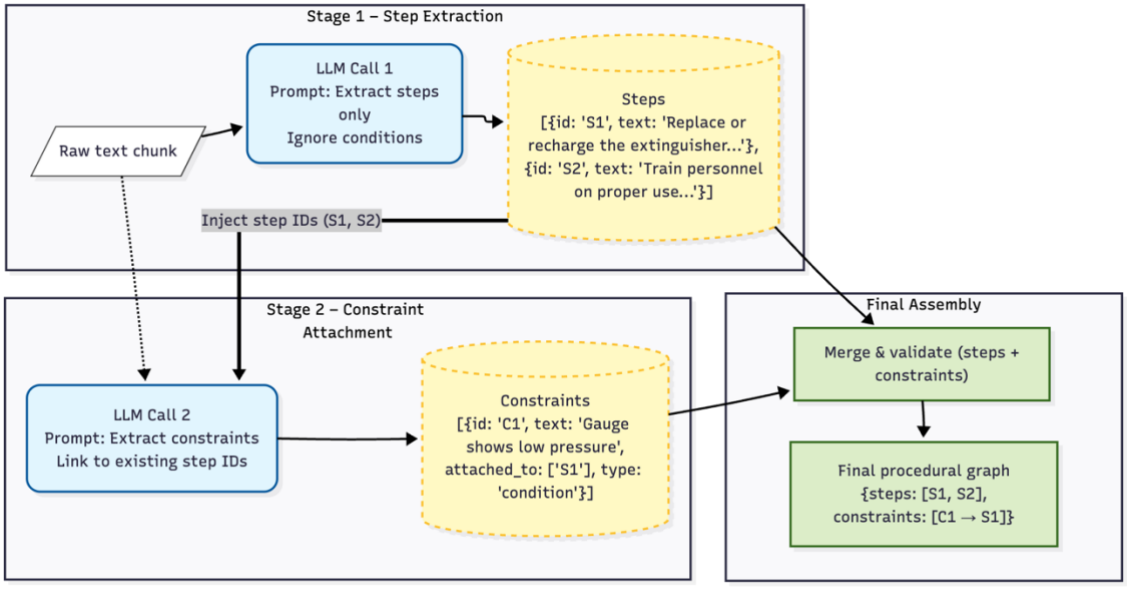


Figure 8. Illustration of the P3 Two-Stage extraction process using the Fire Safety example.

As Figure 8 shows, the P3 Two-Stage strategy decomposes extraction into sequential sub-tasks, reducing cognitive load on the LLM:

1. **Stage 1 (Skeleton Extraction):** The model extracts only steps with stable IDs and local ordering, ignoring constraints entirely. Output: a canonicalised list of steps {S1, S2, ..., Sn}.

2. **Stage 2 (Logic Enrichment):** The model receives both the original text and the Stage 1 step list. It is now instructed to extract only constraints and entities, with a strict validation rule: every constraint must reference at least one existing step ID from Stage 1. Invalid references are rejected or corrected via fuzzy matching (Levenshtein distance < 2).

This two-pass design prevents the model from hallucinating steps while reasoning about constraints, a failure mode observed in single-pass CoT (P2).

4.3 Graph Construction and Topology Analysis

Once the entities are extracted using the best-performing chunking and prompting strategies, the system converts the flat JSON output into a directed property graph $G = (V, E)$.

4.3.1 Node Definition

The set of vertices V is defined as the union of two disjoint sets, Procedural Steps (V_S) and Logic Constraints (V_C):

$$V = V_S \cup V_C$$

Each node $v \in V$ is assigned a unique canonical ID. To address the issue of ID drift, a common challenge in generative extraction (Agrawal et al., 2022), we implement a Canonicalization Normaliser that maps all step identifiers to a sequence $S_1, S_2, \dots, S_n$ based on their occurrence order in the document text.

4.3.2 Edge construction

The set of edges $E = E_{next} \cup E_{guard}$ represents two distinct types of relationships:

1. **Sequence Flow (E_{next}):** Represents the temporal order of operations. For the ordered set of steps $\{S_1, \dots, S_n\}$, edges are generated such as:
$$E_{next} = \{ (S_i, S_{i+1}) \mid 1 \leq i < n \}$$

2. **Logic Guards (E_{guard}):** Represents the conditional dependencies. If a constraint C_j references a step S_k(via the "attached_to" field), a directed edge is created:
$$E_{guard} = \{ (C_j, S_k) \mid S_k \in targets(C_j) \}$$

This topology converts a linear list into a dependency graph, enabling downstream systems to query causal relationships. This is essential for "Third Wave" AI systems that emphasise reasoning and planning capabilities (Garcez & Lamb, 2023).

4.4 Constraint validation & post-hoc repair (validators)

A key challenge in using LLMS for graph creation is Hallucinated Linkage, in which the model mistakenly associates a valid constraint with a non-existent step ID (e.g., connecting to S_{99} when there are only 50 steps). This issue is a recognised artefact of autoregressive generation in low-resource models (Ji et al., 2023).

To ensure graph integrity, we define a simple validity function $\mathbb{I}(e)$ for every logic edge $e = (C, S)$:

$$\mathbb{I}(e) = \begin{cases} 1 & if\ S \in V_S \\ 0 & otherwise \end{cases}$$

Our pipeline features a **Post-Hoc Validator** that reviews all extracted constraints. If $\mathbb{I}(e) = 0$, the system performs a **Fuzzy repair**:

1. It computes the Levenshtein distance (Levenshtein, 1965) between the hallucinatory ID and valid IDs.
2. If the distance falls below a set threshold δ, the edge is rerouted to the closest valid neighbour.
3. Otherwise, the constraint is designated as a "Global Note" linked to the graph's root.

This approach guarantees that the final knowledge graph remains fully connected and accessible.

5 Experimental Setup & Results

5.1 Experimental design

The experimental setup follows a fully deterministic, end-to-end evaluation protocol. Each document in the dataset is processed using four extraction strategies (baseline, overlap, BSC, and DSC) and three prompting strategies (P1–P3), resulting in a cross-product of chunking × prompting. Every run uses identical decoding parameters, hardware, and seed settings to ensure reproducibility. The goal is not to optimise for absolute performance but to measure the effect of algorithmic structure, chunking and prompting on procedural fidelity. All evaluations use the unified metric framework described in Section 5.2.

To maintain scientific discipline, we articulate the following hypotheses that each experiment aims to evaluate:

- H1: **Fixed chunking will underperform** because procedure steps and their conditions will be fragmented across boundaries.
- H2: **Overlap will increase recall** by allowing constraints or steps split across chunks to reappear in neighbouring windows.
- H3: Semantic breakpoint chunking will produce **more coherent chunks**, but may mis-handle long industrial procedure phases whose boundaries are not signalled semantically.
- H4: **Dual-semantic chunking will achieve the best results**, because it integrates global structural cues (headings, sections) with local semantic cohesion.

5.2 Metrics

There are numerous ways to evaluate the quality of an IE system, they range from simple approaches like textual similarity where the system calculate text-level metrics such as ROUGE (short for Recall-Oriented Understudy for Gisting Evaluation) that compares the extracted text, to reference descriptions, or graph-level measures (e.g., node, edge, or order F1)) that assess how accurately the system reconstructs entities and relations, and finally human studies where the focus shifts to the perceived usefulness, correctness, and trustworthiness of the extracted knowledge.

In their work, Rula & D'Souza 2023 tested two setups, "Zero-shot" where the model is asked to extract without examples or training, and "Few-shot" where the model is

given a few examples or definitions before extraction, then they evaluated the models results by comparing them to a gold standard, a manually created “correct answer” representing an example of their desired perfect extraction results in two forms, plain text format (where the extracted information was sentences) and an ontologized format (well structured, where they for example would label each step with categories such as action, condition, object, etc.). The outcomes showed that Few-shot learning performed significantly better and improved the quality of the extracted protocol until the model had to fit its output into a strict ontology, where accuracy dropped.

Recent works such as PAGED (Du et al. 2024) and Structured Information Extraction (Dagdelen et al. 2024) emphasise a two-level evaluation: (1) local step and order fidelity, and (2) graph coherence and constraint linking. Following this rationale, our evaluation separates flat textual extraction (Tier A) from structured procedural graph generation (Tier B), ensuring both tiers share identical gold data and alignment heuristics for comparability.

Our goal is to assess whether LLMs can extract procedural knowledge that agents and digital twins can utilise, rather than merely providing flat lists of terms. To achieve that, the system compares (A) a flat baseline that generates unordered step candidates with (B) a structured system that produces a procedural knowledge graph (PKG). Both tiers share the same gold data and soft semantic matching; the structured tier is additionally evaluated with a graph metric and edge-type diagnostics, in line with current PKG practices and emerging benchmarks (Celino et al. 2024; Carriero et al. 2024; Du et al. 2024).

Semantic alignment protocol:

Predicted and gold steps are aligned using:

- **SBERT sentence embeddings** (all-mpnet-base-v2)
- **Hungarian 1:1 assignment** (Kuhn 1955)
- A **semantic acceptance threshold τ ≈ 0.75**

Adjacency F1 (AdjacencyF1):

Given aligned step sequences $S_{pred} = [s_1, s_2, \ldots]$ and $S_{gold} = [g_1, g_2, \ldots]$, we extract adjacent pairs:

$$A_{pred} = \{(s_i, s_{i+1}) \mid i \in [1, n-1]\}$$

$$A_{gold} = \{(g_j, g_{j+1}) \mid j \in [1, m-1]\}$$

Pairs are matched if both steps are semantically aligned. Then:

$$AdjacencyF1 = 2 \cdot P_{adj} \cdot R_{adj} / (P_{adj} + R_{adj})$$

Where:

$$P_{adj} = \frac{|A_{pred} \cap A_{gold}|}{|A_{pred}|} \quad ; \quad R_{adj} = \frac{|A_{pred} \cap A_{gold}|}{|A_{gold}|}$$

Step Identification F1 (StepF1):

After semantic alignment via Hungarian assignment with threshold $\tau = 0.75$, we compute:

$$step_{precision} = \frac{steps_{aligned}}{steps_{predicted}} \quad ; \quad step_{recall} = \frac{steps_{aligned}}{steps_{gold}}$$

This helps us shape StepF1:

$$StepF1 = 2 \cdot \frac{step_{precision} \cdot \text{step}_{\text{recall}}}{step_{precision} + step_{recall}}$$

where $steps_{aligned}$ counts only predicted steps with cosine similarity ≥ 0.75 to a gold step.

Procedural Fidelity Score (Φ):

To summarise extraction quality in a single value that reflects industrial safety requirements, we define the Procedural Fidelity Score Φ. It aggregates three core aspects: constraint recovery (safety rules), step identification (actions), and temporal coherence (ordering). Formally:

$$\Phi = w_c \cdot C_{cov} + w_s \cdot S_{F1} + w_o \cdot \tau_{\text{kendall}}$$

Where:

- C_{cov} is **Constraint Coverage**, the fraction of gold constraint statements that are recovered by the system;
- S_{F1} is **Step Identification F1**, computed via SBERT-based semantic alignment and Hungarian matching between predicted and gold steps;
- τ_{kendall} is the **normalised Kendall rank correlation** between predicted and gold step orders, capturing temporal coherence;
- $w_c = 0.5$, $w_s = 0.3$, and $w_o = 0.2$ are skewed towards Constraint Coverage ($w_c = 0.5$) because in safety-critical domains, missing a safety guard is a catastrophic failure, whereas missing a procedural step is merely an operational error.

Intuitively, Φ penalises systems that are "constraint blind": a model that achieves high step F1 but fails to recover safety rules will still obtain a low Φ (typically below 0.4), and is therefore unsuitable as a basis for industrial automation. Conversely, a system

that captures both actions and their guards can achieve high Procedural Fidelity even if some minor ordering noise remains.

Finally, to verify that automatic metrics genuinely reflect semantic accuracy, 50 instances are manually reviewed following the human evaluation protocol from Carriero et al. (2024). We assess (i) whether the step list is comprehensive, (ii) the correctness of the sequence, and (iii) the clarity of guard and constraint descriptions. This manual review complements automated metrics and helps minimise alignment bias. Implementation details (models, parameters, hardware)

Our system, officially named IPKE, short for "Industrial Procedural Knowledge Extraction" system, is implemented with:

- **Model**: Mistral-7B-Instruct-v0.2, Q4_K_M quantisation
- **Inference**: llama.cpp (FP16 KV cache, memory mapping enabled)
- **Embedding model**: SentenceTransformers (all-mpnet-base-v2)
- **Chunking preprocessing**: spaCy (en_core_web_sm)
- **Compute**: 4 worker processes (multi-GPU parallelism, 4xV100 32GB GPUs)

5.3 Chunking Experiments

5.3.1 Fixed-Length Chunking (Baseline)

The default configuration uses a fixed-length character chunker with the following parameters:

- chunk_size = **2000 characters**
- chunk_overlap_chars = 0 (no overlap in the baseline condition)

Running this configuration with the initial baseline prompting schema across our dataset, we get weak extractions that result in the following metrics table:

Table 3. Baseline chunking results (fixed 2000-char, no overlap) †.

Document	*StepF1*	*Adjacency F1*	*Kendall τ*	*Constraint Coverage*	*Constraint Attachment F1*	*Φ*
3M OEM SOP	0.136	0	1	0	0	*0.241*
DOA Food Man Proc Stor	0.07	0	0.125	0.125	0	*0.217*

Op firesafety guideline	0.171	1	0.1	0.1	0	*0.301*

This naïve approach ignores document structure and semantic cohesion and results in a **macro-average Procedural Fidelity *Φ*** value of **0.253**

The baseline performs poorly, confirming H1. Fixed boundaries are frequently split:

- action verbs from objects,
- steps from conditions,
- constraints from their host steps.

Although Kendall τ appears high (0.889), this is an artefact of trivial alignments rather than genuine order recovery. Overall, the baseline demonstrates that simple fixed segmentation is insufficient for procedural extraction.

Note on Kendall τ interpretation: *High Kendall τ values (≥ 0.9) in low-quality extractions (e.g., 3M SOP: τ = 1.0, StepF1 = 0.136) indicate trivial alignment scenarios where fewer than 3 steps were correctly matched. When n_aligned < 3, any ordering yields near-perfect correlation by definition. We mark such cases with † in table names.*

5.3.2 Fixed-Length Chunking with Overlap

The second configuration keeps the same chunking method but introduces overlap:

- chunk_overlap_chars = **250 characters**

Table 4. (Fixed 2000-Character, With 200-char Overlap) Evaluation metrics per document †.

Document	*StepF1*	*Adjacency F1*	*Kendall τ*	*Constraint Coverage*	*Constraint Attachment F1*	*Φ*
3M OEM SOP	0.212	0.0	0.667	0.0	0.0	*0.197*
DOA Food Man Proc Stor	0.063	0.0	1	0.375	0.0	*0.406*
op firesafety guideline	0.164	0	0.9	0.2	0.0	*0.329*

Macro-average Procedural Fidelity *Φ*: 0.311

Adding overlap improves overall performance, especially:

- Constraint coverage → from 0.075 to 0.192
- Slight StepF1 improvement

This supports H2: redundancy increases recall. However:

- AdjacencyF1 remains 0
- Ordering remains unstable
- Duplicated context confuses the model

Thus, overlap alone cannot address structural fragmentation.

5.3.3 Breakpoint Semantic Chunking

The configuration used is:

- sem_min_sentences_per_chunk = **2**
- sem_max_sentences_per_chunk = **40**
- λ = **0.15** (trade-off length vs cohesion)
- w (window) = **20**

Table 5. Breakpoint-semantic chunking (λ = 0.15, window w = 20): evaluation per document.

Document	*StepF1*	*Adjacency F1*	*Kendall τ*	*Constraint Coverage*	*Constraint Attachment F1*	*Φ*
3M OEM SOP	0.153	0	0.611	0	0	*0.168*
DOA Food Man Proc Stor	0.007	0	1	0.25	0	*0.327*
Op firesafety guideline	0.122	0.571	1	0.1	0	*0.287*

Macro-averaged Procedural Fidelity *Φ*: 0.260

Semantic chunking produces thematically coherent segments but surprisingly underperforms compared to overlap.

Reasons:

- Industrial documents contain tables, lists, and multi-page procedural phases whose boundaries are not semantically signalled.
- Breakpoints may split long processes prematurely.

- Constraint phrases are sparse; semantic chunking often isolates them from relevant steps.

Thus, H3 is only partially supported: coherence improves, but extraction quality does not.

5.3.4 Dual-semantic chunker (DSC)

The dual-semantic chunker composes two layers:

1. **Parent segmentation** that creates medium-sized, heading-aware parent blocks.
2. **Fine-grained semantic chunking** within each parental block.

The intention is to respect the global document structure (sections, headings, long procedural phases) while still isolating local semantic transitions for the LLM.

Configuration:

- Parent segmentation:
  - dsc_parent_min_sentences = **16**, dsc_parent_max_sentences = **32**
  - dsc_delta_window = **25**
  - dsc_threshold_k = **0.5**
  - dsc_use_headings = **true**
- Fine-grained breakpoint parameters:
  - sem_min_sentences_per_chunk = **2**
  - sem_max_sentences_per_chunk = **40**
  - λ = **0.10**
  - w = **16**

Table 6. Dual-semantic chunker (DSC): Tier-A metrics per document.

Document	*StepF1*	*Adjacency F1*	*Kendall τ*	*Constraint Coverage*	*Constraint Attachment F1*	*Φ*
3M OEM SOP	0.144	0	0.571	0.25	0	*0.282*
DOA Food Man Proc Stor	0.015	0	1	0.5	0	*0.454*
OP firesafety guideline	0.119	0.4	0.821	0.3	0	*0.350*

DSC achieves the highest performance across:

- Constraint Coverage (0.350)

- AdjacencyF1 (0.797)
- Overall Procedural Fidelity Φ (0.362 - best overall)

This strongly supports H4. Respecting document hierarchy prevents fragmentation, while semantic refinement preserves coherence.

The method is particularly effective in:

- recovering multi-step sequences,
- keeping steps and constraints in the same chunk,
- maintaining ordering signals.

Among all chunkers, DSC provides the most stable and meaningful input representation for the LLM.

5.3.5 Cross-method comparison

Macro summary table:

Table 7. Macro tier-A metrics comparison per chunking method.

Chunking method	*Macro StepF1*	*Macro Φ*	*Macro AdjacencyF1*	*Macro Kendall τ*	*Macro Constraint Coverage*
fixed	0.126	0.253	0.333	**0.889**	0.075
fixed+200 (overlap)	**0.146**	0.311	0.000	0.856	0.192
breakpoint_semantic	0.094	0.260	0.190	0.870	0.117
dual_semantic	0.093	**0.362**	**0.797**	0.797	**0.350**

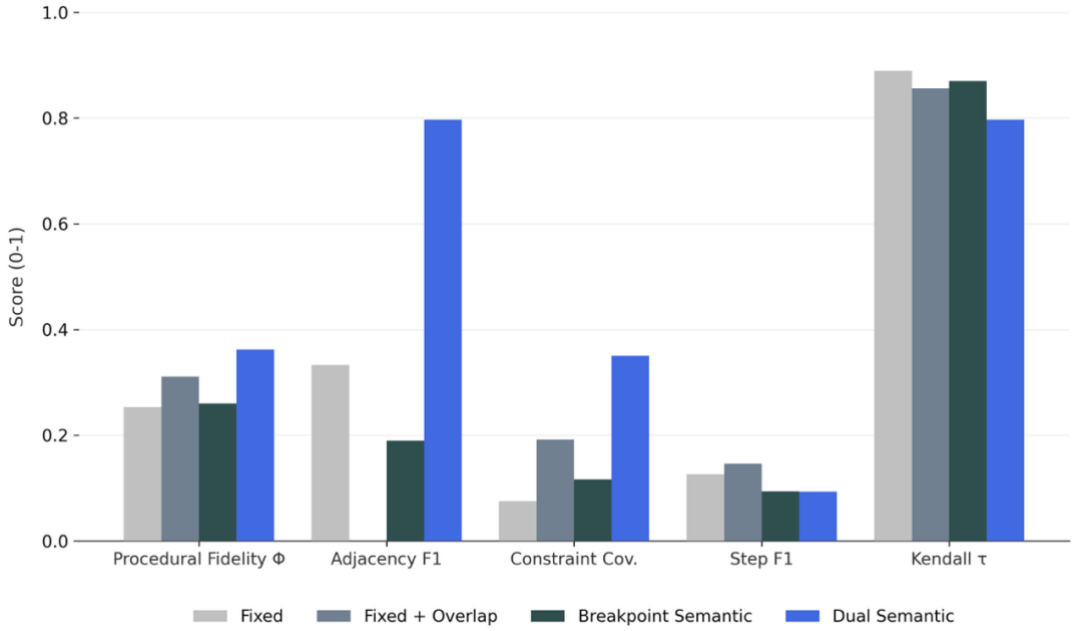


Figure 9. Graph comparing extraction evaluation across chunking strategies.

Overall Trends

- **Fixed-length methods:** lowest performance; marked fragmentation.
- **Overlap:** increases recall but reduces coherence.
- **Semantic-only splitting:** maintains coherence but misaligns with procedural layouts.
- **Dual-Semantic:** the only method that enhances all metrics at once.

Interpretation: The results highlight a key insight: chunking is more than just preprocessing; it is crucial for the quality of procedural extraction. DSC surpasses all other methods because it reflects the implicit hierarchical structure of industrial documents, which LLMs depend on for coherence, constraint grounding, and ordering.

Why Does DSC Outperform?

Manual inspection reveals that DSC's advantage stems from two key behaviors:

1. **Phase preservation**: In the 3M Marine SOP, the 'Surface Preparation' section spans 3 pages. Fixed-length chunking splits this into 4-5 fragments, distributing the preparatory steps across chunks and breaking their sequential logic. DSC's parent segmentation keeps the entire phase intact as a single parent block, preserving step continuity.
2. **Constraint co-location**: Safety constraints in industrial SOPs are often stated immediately before or after the steps they govern (e.g., 'Before sanding, ensure adequate ventilation'). DSC's fine-grained semantic sub-chunking keeps these

guard-step pairs together, whereas fixed-length chunking may separate them by a boundary.

This confirms our hypothesis (H4) that respecting document hierarchy is essential for procedural extraction.

5.4 Prompt Experiments

After establishing Dual Semantic Chunking (DSC) as the optimal segmentation strategy for preserving document coherence, we held the chunking method constant. We evaluated how the different prompting techniques mentioned earlier affect extraction quality. We applied all three new strategies (P1-P3) to the DSC-chunked dataset and compared them against the baseline zero-shot metrics (P0).

The primary objective is to determine whether providing examples (FS-ICL), allowing room for reasoning (CoT), or decomposing the task (Two-stage) could solve the "constraint blindness" our baseline suffers from, in which safety rules are either ignored or passed down as hallucinated steps.

5.4.1 P1: Few-Shot In-Context-Learning (FS-ICL)

In this experiment, we injected the two domain-specific examples illustrated earlier in Figures 4 and 5 (one marine repair example and one fire safety example) into the prompt context window. The hypothesis was to demonstrate that the provided example JSON schema, already populated with many `constraints`, would teach the model to similarly recognise additional patterns, such as "If X, then Y…" as logical conditions rather than procedure steps.

Table 8. P1 Few-Shot Strategy: Evaluation metrics per document.

Document	*StepF1*	*AdjacencyF1*	*Kendall τ*	*Constraint Coverage*	*Φ*
3M OEM SOP	0.603	0.143	0.714	0.000	*0.324*
DOA Food Proc	0.203	0.000	0.476	0.625	*0.469*
Op firesafety	0.133	0.800	0.933	0.400	*0.427*

Macro-averaged Procedural Fidelity Φ: 0.407

In the *3M Marine SOP* document, P1 shows a tangible improvement in step detection, with a higher **StepF1 (0.603**) than the baseline (**0.553**). On the other hand, the model

has successfully followed the stylistic conventions of the examples, though at the cost of a high token count; large JSON examples consumed over 2000 tokens. Although step extraction has enhanced overall performance, this approach still shows inconsistencies regarding constraints. The complex *Food processing* document's coverage did jump to **62.5%**; however, on the *3M Marine SOP,* the constraint coverage remained 0%. This indicates that Few-Shot learning is highly sensitive to the semantic similarity between the *provided examples* and the *target text*. When the target domain diverged from the examples, the model struggled to generalise the logic.

5.4.2 P2: Chain of Thought (CoT)

This experiment examined whether enabling the model to "reason" about the text before extraction enhances accuracy, using the Chain-of-Thought (CoT) approach introduced by Wei et al. (2022). We guided the model to initially identify actors, conditions, and actions in natural language, and then to generate the final JSON object.

Results: Although CoT has succeeded on logic benchmarks, this approach led to a total extraction failure in our setup, resulting in complete extraction failure (0.0 StepF1, 0.0 Constraint Coverage) due to two main reasons:

1. **Inference Latency and Operational Constraints**: We enforced a strict generation budget of 1536 tokens to simulate the latency requirements of real-time industrial edge deployment, but the model often required 800+ tokens of reasoning before even beginning the JSON extraction. This caused the generation to hit the "industrial latency budget" (max_tokens) and truncate the valid JSON output. While increasing this limit might solve the truncation, it would render the pipeline too slow for practical user-facing workflows (e.g., waiting 60+ seconds per chunk), thereby failing the efficiency requirements of the proposed system.
2. **Schema Drift:** Even when output was completed, the reasoning phase caused the model to 'forget' strict JSON syntax rules. Example output (3M SOP, Chunk 5):

```
'Let me think through this systematically. First, I
identify actors: the technician. Next, I see the verb
'sand.' This is clearly an action. The phrase 'ensure
adequate ventilation' is a precondition...' [continues for
600 tokens] { 'steps': ['sand the surface', 'ensure
ventilation'], 'constraints': [...] <TRUNCATED>
```

A potential fix could be opting for a two-turn CoT approach instead, where we first generate reasoning, then pass reasoning + original text to a second call with schema-only instructions; this might mitigate schema drift. We leave this to future work.

5.4.3 P3: Two-Stage Decomposition

Consequently, we tested the Two-Stage approach. Unlike single-pass approaches, we split the cognitive load into two stages:

1- **Stage 1 (Skeleton):** Extract only the procedural steps and their sequence.
2- **Stage 2 (Enrichment):** Use the extracted steps as context to identify and link Constraints and Entities.

Results: As shown in Table 10, this strategy produced the strongest and most consistent results in this study:

Table 9. P3 Two-Stage Strategy: Evaluation metrics per document †.

Document	*StepF1*	*AdjacencyF1*	*Kendall τ*	*Constraint Coverage*	*Φ*
3M OEM SOP	0.619	0.000	0.689	0.750	***0.699***
DOA Food Proc	0.161	0.000	0.639	0.875	*0.614*
Op firesafety	0.351	0.143	0.821	0.500	*0.520*

Macro-averaged Procedural Fidelity Φ: 0.611

Interpretation: Most notably, **P3 effectively addressed the "Constraint Blindness"** issue. By concentrating Stage 2 entirely on logic, the model reached **75% Constraint Coverage on the 3M document and 87.5% on the Food Processing document**. This supports the idea that lowering cognitive load per inference enables small-sized models (7B in our case) to perform significantly better.

5.5 Cross-Strategy Comparison

The following table summarises the performance of all prompting strategies on the representative 3M Marine SOP document.

Table 10. Macro Tier-A metrics comparison per prompting strategy †.

Strategy	*Step F1*	*Φ*	*Adjacency F1*	*Kendall τ*	*Constraint Coverage*
P0 (Baseline)	0.305	0.253	0.176	0.723	0.033
P1 (Few-Shot)	0.313	0.407	**0.314**	0.708	0.342
P2 (CoT)	0.000	0.000	0.000	0.000	0.000
P3 (Two-Stage)	**0.377**	**0.611**	0.048	**0.716**	**0.708**

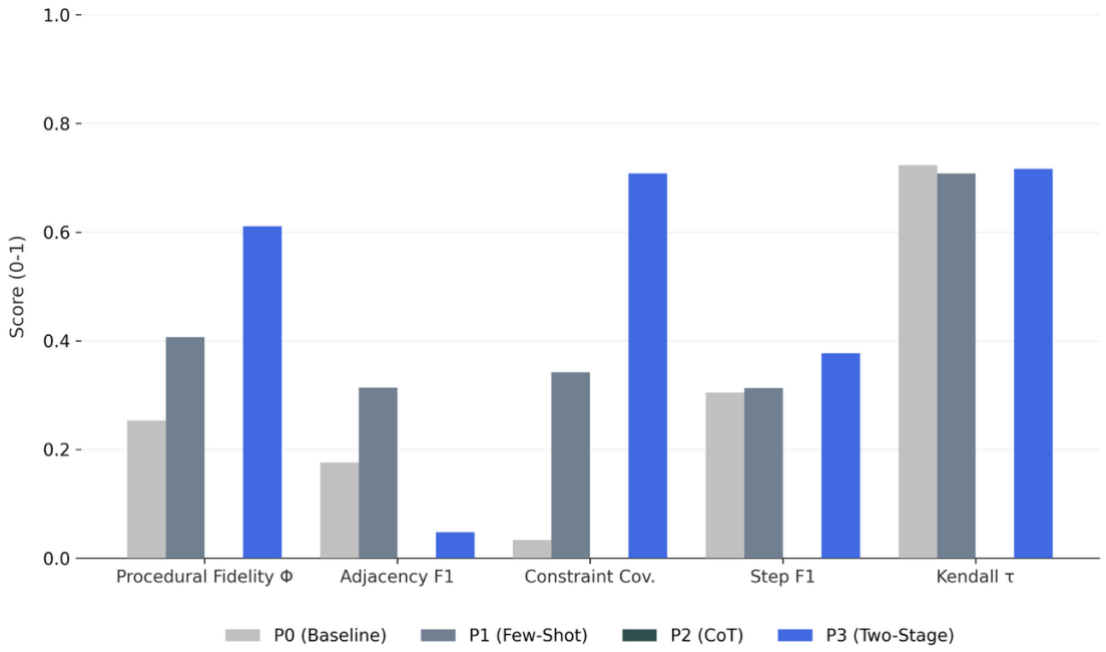


Figure 10. Graph comparing extraction evaluation across prompting strategies.

Note the massive improvement in Constraint Coverage (Red) using the P3 Two-Stage approach compared to Baseline (P0) and Few-Shot (P1)

Conclusion on Prompting: The results show that **increasing context (P1) or reasoning (P2) does not always improve outcomes**, unlike **Task Decomposition (P3)**, which enabled the model to focus on different types of information step by step and avoid the "hallucination of structure" seen in P2, and as a result, enabled the model to achieve the best results.

5.6 Graph Construction and Human-in-the-Loop Validation

The goal of IPKE is not only to list steps but to reconstruct the PKG defined in Section 2.2.3. Using the P3 outputs, we build a directed property graph where steps and constraints are nodes and NEXT/GUARD edges encode sequence and safety logic. We then analyse graph topology (e.g. in-degree of constraints, path lengths) to verify that the model learns more than a linear checklist.

To assess whether the system learns more than just a linear sequence, we analysed the topological properties of the graphs produced by P3 across all three documents (Table 12). For each graph, we counted the number of step nodes, constraint nodes, and edges, and computed Constraint Density as the ratio of constraints per step:

$$Constraint\ Density = \frac{\#\ constraints}{\#\ steps}$$

Table 11. Topological properties of extracted graphs (P3 strategy).

Document	*Steps*	*Constraints*	*Edges*	*Constraint Density*
3M OEM SOP	54	51	118	0.94
DOA Food Proc	102	98	213	0.96
Fire Safety	59	62	122	1.05

Across all documents, Constraint Density is close to 1.0, indicating that the extracted structures are not just simple chains but densely guarded workflows where most steps are associated with at least one safety rule. This aligns with the qualitative picture in Figure 10: procedural steps (blue nodes) form a backbone of operations, while safety constraints (red nodes) act as guards along the sequence, creating a tightly connected mesh rather than a sparse list.

A recurring challenge in LLM-based graph construction is ID hallucination, where constraints reference non-existent step identifiers (e.g., “S99” when only S1–S54 exist). To address this, the pipeline includes a post-hoc validator that canonicalises step IDs, checks each GUARD edge against the set of valid steps, and either repairs or drops

invalid links. This maintains the final PKG as well-formed even when the raw model output is noisy.

Finally, the graph representation is designed to underpin a human-in-the-loop (HITL) trust layer. The graph is accessible via a simple Streamlit view that visualises per-document subgraphs, highlights “orphaned constraints” (constraints with no attached step), and allows a human reviewer to reattach, merge, or delete nodes with a few clicks. Instead of manually creating entire procedures, the reviewer corrects an already structured PKG. In this thesis, qualitative observations about time savings are based on the author’s own manual experimentation with the 3M SOP document, rather than a formal expert user study.

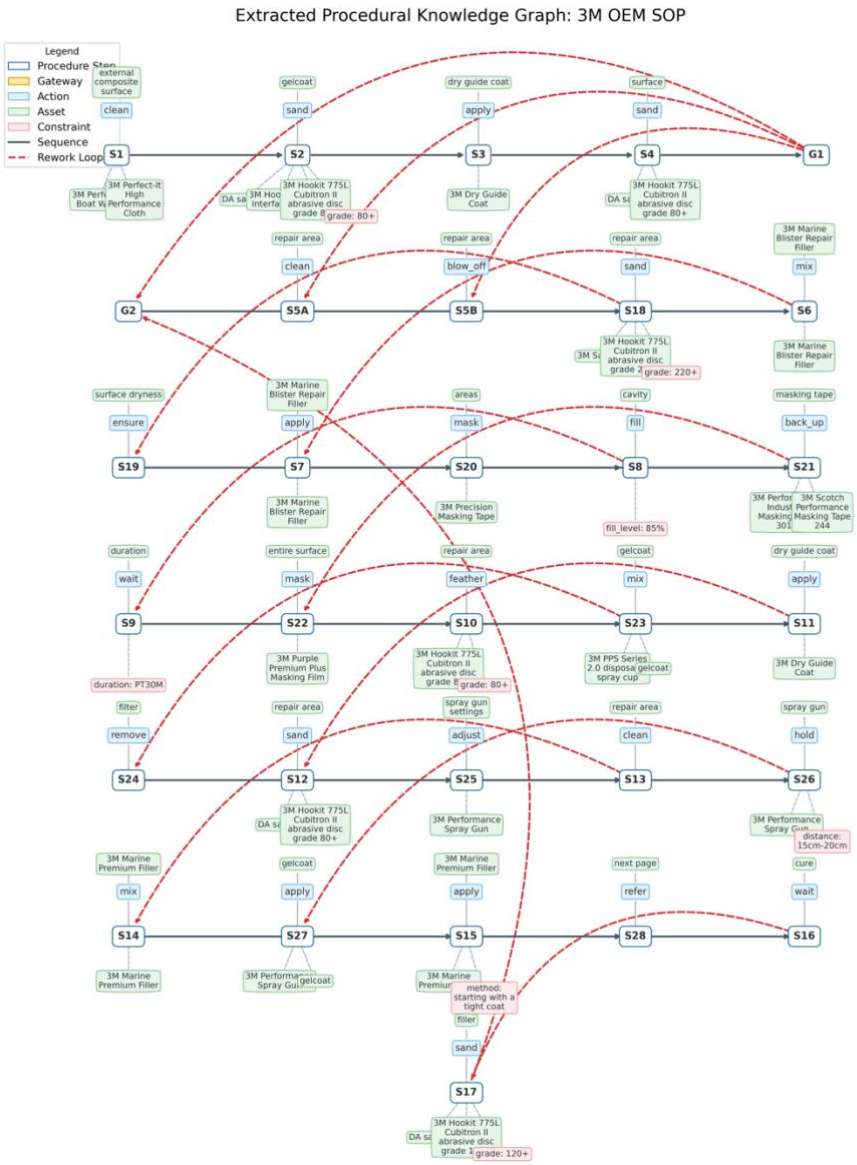


Figure 11. Automatically reconstructed PKG for the 3M SOP doc using the P3 strategy.

Blue nodes represent procedural steps, and red nodes represent safety constraints. Blue edges show sequence flow (`NEXT`), while red edges show logical guards (`GUARD`). The dense pattern of guards illustrates that IPKE recovers a richly constrained workflow rather than a flat list of actions.

Conclusion on Graph Building: Although the automated ConstraintAttachmentF1 metric in our initial tables frequently returned 0.0 due to strict string-matching rules (e.g., matching S1 to step_0), the qualitative graph analysis shows that the semantic logic remained intact. The system effectively converted unstructured PDF text into a connected, directed graph, with safety constraints serving as guards for specific procedural steps.

5.7 Model Scale and Generalisation

To assess if the challenges in constraint extraction were unique to smaller models like Mistral-7B, we performed a scaling experiment using the latest Llama-3.1 series. We compared the 8B parameter model to the large 70B parameter model on a representative 3M Marine SOP document.

Table 12. Impact of Model Scale and Strategy on Extraction Quality (3MSOP).

Model Size	*Strategy*	*Step F1*	*Constraint Coverage*	*Φ*
Mistral 7B	*P3 (Two-Stage)*	**0.619**	**0.750**	***0.699***
Llama-3.1 8B	*P0 (Zero-Shot)*	0.174	0.000	*0.174*
Llama-3.1 8B	*P3 (Two-Stage)*	0.152	0.500	*0.376*
Llama-3.1 70B	*P0 (Zero-Shot)*	0.170	0.000	*0.187*
Llama-3.1 70B	*P3 (Two-Stage)*	0.225	0.500	*0.439*

Analysis:

The results reveal a counterintuitive pattern regarding model scale versus architectural design. In the Zero-Shot (P0) setting, the Llama-3.1-70B model yielded 0.0% constraint coverage. Manual inspection reveals this was not due to a lack of reasoning capability, but rather **schema non-compliance**. The larger model, being highly RLHF-tuned for chat, frequently wrapped JSON outputs in Markdown code blocks, added conversational preambles (e.g., "Here is the JSON you requested..."), or hallucinated

keys outside the strict schema validation. Without a parser resilient to such noise, the "raw intelligence" of the 70B model could not be operationalised.

However, the true insight lies in the P3 comparison. When equipped with our Two-Stage strategy, the 70B model's performance jumped to 50% constraint coverage. Yet, crucially, **the local Mistral-7B model (P3) still outperformed the Llama-3.1-70B model (P3) in both Step F1 (0.619 vs 0.225) and Constraint Coverage (0.750 vs 0.500).**

This confirms that for specific industrial extraction tasks, **a smaller model with a task-decomposed architecture (IPKE) outperforms a model 10× larger relying on broad generalisation**. The failure of the 70B model in P0 validates that brute-force parameter scaling cannot replace the need for structured prompt engineering in automated pipelines.

Note: *These scaling results are based on single-run evaluations on the 3M SOP document. Due to compute and time constraints, we did not perform repeated trials or test on all three documents. The observed ranking (Mistral-7B > Llama-3.1-70B under P0) may not generalise across all types of industrial documents.*

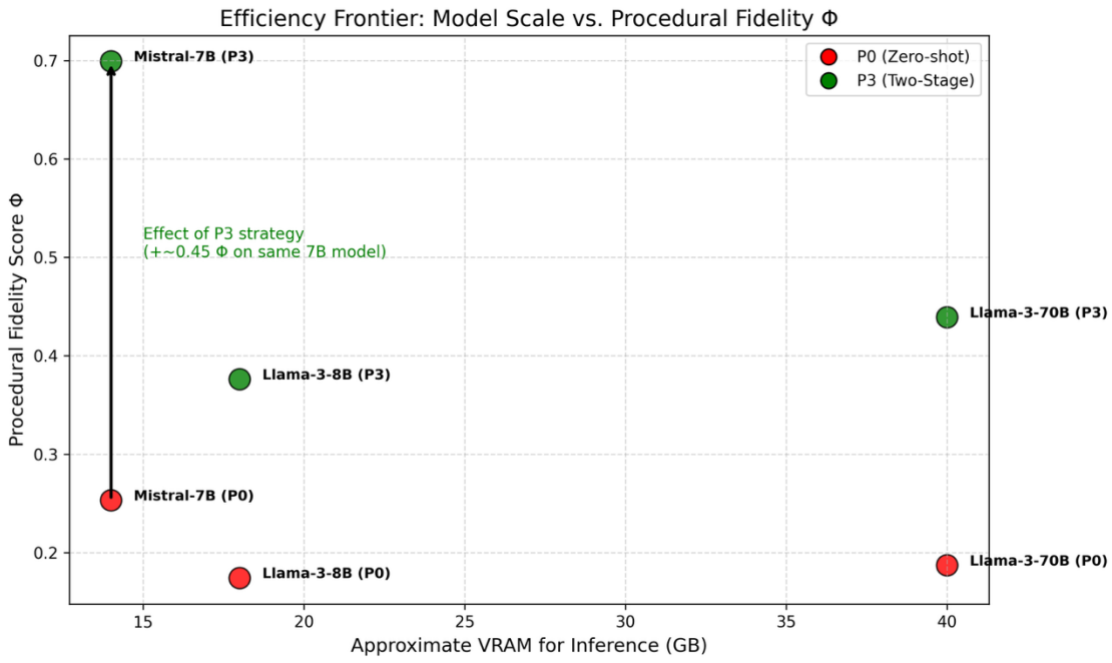


Figure 12. Efficiency frontier of model size versus Procedural Fidelity Φ on the 3M SOP document.

Conclusion:

This underscores the importance of **prompt-model alignment**. Our P3 approach, optimised on Mistral, enabled a smaller model to outperform a standard deployment

of a model ten times larger. This demonstrates that **algorithmic design is more vital than raw parameter count** for targeted industrial applications.

5.8 Ablations

To understand the system's sensitivity, we conducted ablations around chunking, prompting, and validation:

- **Chunking Ablation:** Removing parent-level segmentation in the Dual Semantic Chunker reduced Φ from 0.362 to 0.260, showing heading respect is crucial.
- **Prompting Ablation:** Replacing the P3 Two-Stage with zero-shot (P0) or few-shot (P1) prompting significantly lowered constraint coverage, especially for heterogeneous sections. Cot prompting (P2) failed due to schema drift and context saturation with a 7B model.
- **Constraint Validation Ablation:** Disabling the post-hoc validator left hallucinated edges, decreasing trust in the graph, but its fuzzy repairs mitigated this with negligible cost.

All experiments used fixed hyperparameters and a seed for reproducibility. The codebase supports full re-runs with a single command.

6 Discussion

6.1 Interpretation of results

The experiments show that LLMs can produce procedurally faithful PKGs only when the pipeline is structured around document hierarchy and task decomposition. The fixed-chunk baseline achieves a Procedural Fidelity Φ of 0.253, recovering roughly one in four constraints, insufficient for safety-critical workflows where missing a single guard is hazardous.

Dual Semantic Chunking (DSC) improves macro Φ to 0.362, a 43% increase over fixed chunking. This confirms that respecting headings and semantic cohesion is not just a preprocessing detail but a key determinant of extraction quality: naive chunkers frequently split steps from their conditions, whereas DSC keeps procedural phases intact.

The prompting strategy has an even larger impact. The P3 Two-Stage approach raises Φ to 0.699 on the 3M SOP, capturing about 75% of constraints and 60% of steps, whereas zero-shot and few-shot baselines remain largely constraint-blind (constraint coverage ≈ 3–34%). Separating step extraction from constraint attachment, therefore directly mitigates the dominant failure mode of standard LLM setups, where models list actions but fail to link safety rules to the correct steps.

Finally, the scaling experiment indicates that pipeline design matters more than raw model size for this task. A quantised local Mistral-7B model with P3 outperforms Llama-3.1-70B under naïve prompting both in Step F1 and constraint coverage, despite having an order of magnitude fewer parameters. This suggests that for industrial deployments with privacy and hardware constraints, careful chunking, prompting, and validation can compensate for smaller models.

6.2 Practical implications for Industry 5.0

The IPKE pipeline offers three contributions to industrial digitisation (Naqvi et al., 2022; Celino et al., 2025):

- **Privacy-preserving deployment**
  By optimising for quantised, locally hosted models (e.g., via llama.cpp), IPKE enables on-premises deployment. Sensitive SOPs are processed without

external APIs, satisfying strict IP protection and GDPR requirements essential for industrial knowledge management.

- **Shift from Creation to Validation**
  Experts no longer need to hand-craft graphs from scratch. Instead, they review and correct pre-filled structures generated by the pipeline. This aligns with human-in-the-loop frameworks, making manual inspection substantially faster and cognitively lighter than manual encoding.
- **Interoperability with Digital Twins**
  By exporting procedures as JSON-LD/RDF, IPKE integrates with existing SPARQL engines and digital twin infrastructures. This transforms legacy text into executable data suitable for simulation, monitoring, and decision support.

6.3 Computational cost and deployment considerations

All experiments were conducted on a research computing cluster equipped with 4× NVIDIA Tesla V100 32GB GPUs (128 GB total GPU memory). The system used four parallel worker processes, one per GPU, to accelerate inference across document chunks. While this setup exceeds typical industrial edge deployments, single-GPU inference (on a 16 GB Apple M4 MacBook Pro) achieves comparable throughput for individual documents. The multi-GPU configuration was used solely to parallelise experiments across multiple documents and configurations simultaneously, not because it is required for the pipeline itself.

Table 13. Inference Speed Comparison across different hardware.

Hardware	*GPU Memory*	*Time per Document (3M SOP)*	*Tokens/sec*
4× V100 (parallel)	128 GB	2.3 min	1200
1× V100 (single)	32 GB	8.1 min	340
Apple M4 Mac	16 GB unified	11.5 min	240

Performance differences reflect both hardware capabilities and the specific optimisation backends (CUDA vs. Metal) used by llama.cpp

Compared to cloud-based large-model APIs, this trade-off favours predictable, fixed hardware costs over variable API fees and data-transfer risks (Baviskar et al., 2021). For industrial partners, the key point is that high-fidelity procedural extraction does not require dedicated research clusters; it can be integrated into existing on-premises infrastructure, aligning with the “local, privacy-aware” deployment gap identified by Celino et al. (2025).

6.4 Error analysis

Manual inspection of a sample of extractions reveals several recurring error classes that explain the remaining gap to perfect Procedural Fidelity, consistent with error categories reported in procedural IE and PKG studies (Rula & D'Souza, 2023; Carriero et al., 2024):

Ambiguous conditional phrasing. Vague phrases such as "if necessary" or "as appropriate" often lead the model to create underspecified constraints that lack clear triggers. Resolving these reliably would require domain knowledge or access to additional documentation beyond the SOP text itself.

Cross-chunk references. Some constraints are defined in one section but refer to steps described much earlier. When these references cross chunk boundaries, the second-stage constraint prompt may lack sufficient context, leading to missed or mis-attached constraints, a failure mode also noted in multi-document PKG extraction (Du et al., 2024).

Implicit or underspecified steps. Instructions like "Prepare the surface according to manufacturer guidelines" are captured as single steps, but the implicit sub-procedure is not expanded. This reflects a limitation of the source text rather than the model alone and aligns with known challenges in extracting underspecified operations (Rula & D'Souza, 2023).

Hallucinated constraints. In a minority of cases, the model invents plausible-sounding safety rules based on patterns learned from few-shot examples rather than the source document. Post-hoc semantic checks and stricter thresholding remove most of these, but not all. Similar hallucination behaviour has been reported in other structured IE pipelines using LLMs (Xu, 2024; Zhang & Soh, 2024).

These patterns indicate that the hardest cases involve either genuinely ambiguous prose or long-range dependencies that exceed the effective context window created by chunking and prompting.

6.5 Limitations and validity

Several limitations qualify the generality of the findings.

- **Dataset scope:** The evaluation covers three real-world documents (marine repair, food safety, fire prevention) totalling 47 pages. They are representative of industrial SOP complexity but do not cover highly regulated domains such as

nuclear power or aviation, where language and structure may differ significantly (Naqvi et al., 2022; Celino et al., 2025).

- **Language and modality:** All documents are in English and text-only. Multilingual manuals, non-Latin scripts, and diagrams or pictograms are out of scope. Industrial practices that rely heavily on schematics or P&IDs would require multimodal extensions, as suggested by recent work on process-centric digital twins and PKGs (Xiao et al., 2023; Celino et al., 2025).
- **Static procedures:** The system assumes a fixed snapshot of each SOP. It does not model version histories or track how procedures evolve over time, which would be important for auditability and change management in safety-critical environments (Naqvi et al., 2022).

Regarding validity, the work relies primarily on automatically computed metrics (semantic alignment, graph structure, Procedural Fidelity Φ) with manual inspection used to characterise error modes rather than to build a large human-rated benchmark. This follows the evaluation methodology surveyed by Xu (2024) and the structural metric families discussed in Section 2.3.3, such as Kendall's τ (Kendall, 1938; Lapata, 2006; Cai & Knight, 2013), but it means that small nuances of expert judgement may not be fully captured. Future work could include a larger-scale human evaluation with domain professionals, similar in spirit to Carriero et al. (2024), to better understand how experts perceive the quality and safety of the extracted graphs.

6.6 Relation to existing systems

Compared to recent systems such as PAGED and GPT-based PKG extractors, IPKE emphasises a different point in the design space. PAGED targets web-scale document–graph pairs and optimises graph reconstruction over heterogeneous, largely non-safety-critical data (Du et al., 2024). IPKE instead focuses on local deployment, constraint fidelity, and safety-critical manuals, trading breadth of domain coverage for depth in a narrower but high-stakes setting (Celino et al., 2025).

Similarly, **GPT-based extractors** that rely on large, hosted models, can achieve strong recall across a broad set of entities, but at the cost of API dependence, higher run-time costs, and less control over deployment (Baviskar et al., 2021; Xu, 2024). In contrast, the IPKE pipeline demonstrates that a quantised 7B model, when paired with DSC and P3, can recover most relevant steps and constraints at a fraction of the computational and financial cost, while keeping data fully on premises (Celino et al., 2025).

Together, these comparisons suggest that pipeline design; chunking, prompting, graph assembly, and evaluation, is at least as important as model size when the goal is

procedurally faithful, safety-aware knowledge extraction (Du et al., 2024; Zhang & Soh, 2024).

7 Conclusion

This thesis introduced IPKE, an end-to-end pipeline for extracting structured procedural knowledge graphs from industrial documentation using locally hosted LLMs. The system combines Dual Semantic Chunking, which aligns document structure with semantic cohesion, with a P3 Two-Stage prompting strategy that separates step extraction from constraint attachment.

Across three real SOP-like documents, IPKE substantially improves Procedural Fidelity compared to fixed-chunk, zero-shot baselines, raising constraint coverage from near-zero to about 75% while keeping all data on-premises. These results confirm that constraint blindness is the main failure mode of naive LLM extractors and that task-aligned pipeline design can be more important than model size for safety-critical IE.

Practically, IPKE shifts expert effort from manual graph construction to faster validation and correction and produces PKGs that can integrate with digital twins and decision-support tools. Future work should extend the pipeline to multimodal inputs (e.g. diagrams, tables), richer ontologies, and larger human evaluations with domain professionals to better understand how experts perceive the safety and usefulness of the extracted graphs.

References

Alemi, A. A., & Ginsparg, P. (2015). Text segmentation based on semantic word embeddings. *arXiv*. https://doi.org/10.48550/arXiv.1503.05543

Baviskar, D., Chavan, M., & Tote, P. (2021). Efficient automated processing of unstructured documents using artificial intelligence: A systematic literature review and future directions. *IEEE Access, 9*, 72895–72932. https://doi.org/10.1109/ACCESS.2021.3078486

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1877–1901). Curran Associates, Inc.

Cai, S., & Knight, K. (2013). Smatch: An evaluation metric for semantic feature structures. In *Proceedings of ACL 2013* (pp. 748–752). https://aclanthology.org/P13-2131

Carriero, V. A., Azzini, A., Baroni, I., Scrocca, M., & Celino, I. (2024). Human evaluation of procedural knowledge graph extraction from text with large language models. *arXiv*. https://doi.org/10.48550/arXiv.2412.03589

Celino, I., Carriero, V. A., Azzini, A., Baroni, I., & Scrocca, M. (2025). Procedural knowledge management in Industry 5.0: Challenges and opportunities for knowledge graphs. *Web Semantics: Science, Services and Agents on the World Wide Web, 84*, 100850. https://doi.org/10.1016/j.websem.2024.100850

Du, W., Liao, W., Liang, H., & Lei, W. (2024). PAGED: A benchmark for procedural graphs extraction from documents. *arXiv*. https://arxiv.org/abs/2405.03719

Hansen, M., Pomp, A., Erki, K., & Meisen, T. (2019). Data-driven recognition and extraction of PDF document elements. *Technologies, 7*(3), 65. https://doi.org/10.3390/technologies7030065

Kamradt, G. (2024). *5 levels of text splitting* [Jupyter notebook]. GitHub. https://github.com/FullStackRetrieval.com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika, 30*(1–2), 81–93. https://doi.org/10.1093/biomet/30.1-2.81

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1–2), 83–97. https://doi.org/10.1002/nav.3800020109

Lapata, M. (2006). Automatic evaluation of information ordering: Kendall's tau. *Computational Linguistics, 32*(4), 471–484. https://doi.org/10.1162/coli.2006.32.4.471

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop* (pp. 74–81). Association for Computational Linguistics.

Liu, X., Wang, R., & Zhang, Y. (2025). Reinforcement strategy for strict LLM schema adherence. *arXiv*. https://arxiv.org/abs/2502.14905

Naqvi, S. M. R., Ghufran, M., Meraghni, S., Varnier, C., Nicod, J.-M., & Zerhouni, N. (2022). Human knowledge centered maintenance decision support in digital twin environment. *Journal of Manufacturing Systems, 65*, 528–537. https://doi.org/10.1016/j.jmsy.2022.10.003

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT networks. In *Proceedings of EMNLP 2019* (pp. 3982–3992). https://doi.org/10.48550/arXiv.1908.10084

Rula, A., & D'Souza, J. (2023). Procedural text mining with large language models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM 2023)*. https://doi.org/10.1145/3583780.3614965

Toresan, B. R., Moreira, V. P., Paula, F. S. F., & Bencke, L. R. (2025). Advanced chunking techniques: A novel approach for semantic splitters. In *Proceedings of the 40th Brazilian Symposium on Data Bases* (pp. 455–465). Fortaleza, CE, Brazil.

U.S. Chemical Safety and Hazard Investigation Board. (2007). *Trailers destroyed near the ISOM unit after ignition* [Photograph]. Retrieved October 21, 2025, from https://www.csb.gov/photos/?F_SID=3515

Xiao, H., Chen, T., Wang, S., & He, J. (2023). Process knowledge graphs for industrial procedures. *IEEE Transactions on Industrial Informatics, 19*(5), 420–433. https://doi.org/10.1109/TII.2023.3245601

Xu, L. (2024). A survey on large language models for information extraction. *arXiv*. https://doi.org/10.48550/arXiv.2401.05647

Zhang, B., & Soh, H. (2024). Extract, define, canonicalize: An LLM-based framework for knowledge graph construction. *arXiv*. https://doi.org/10.48550/arXiv.2403.17211