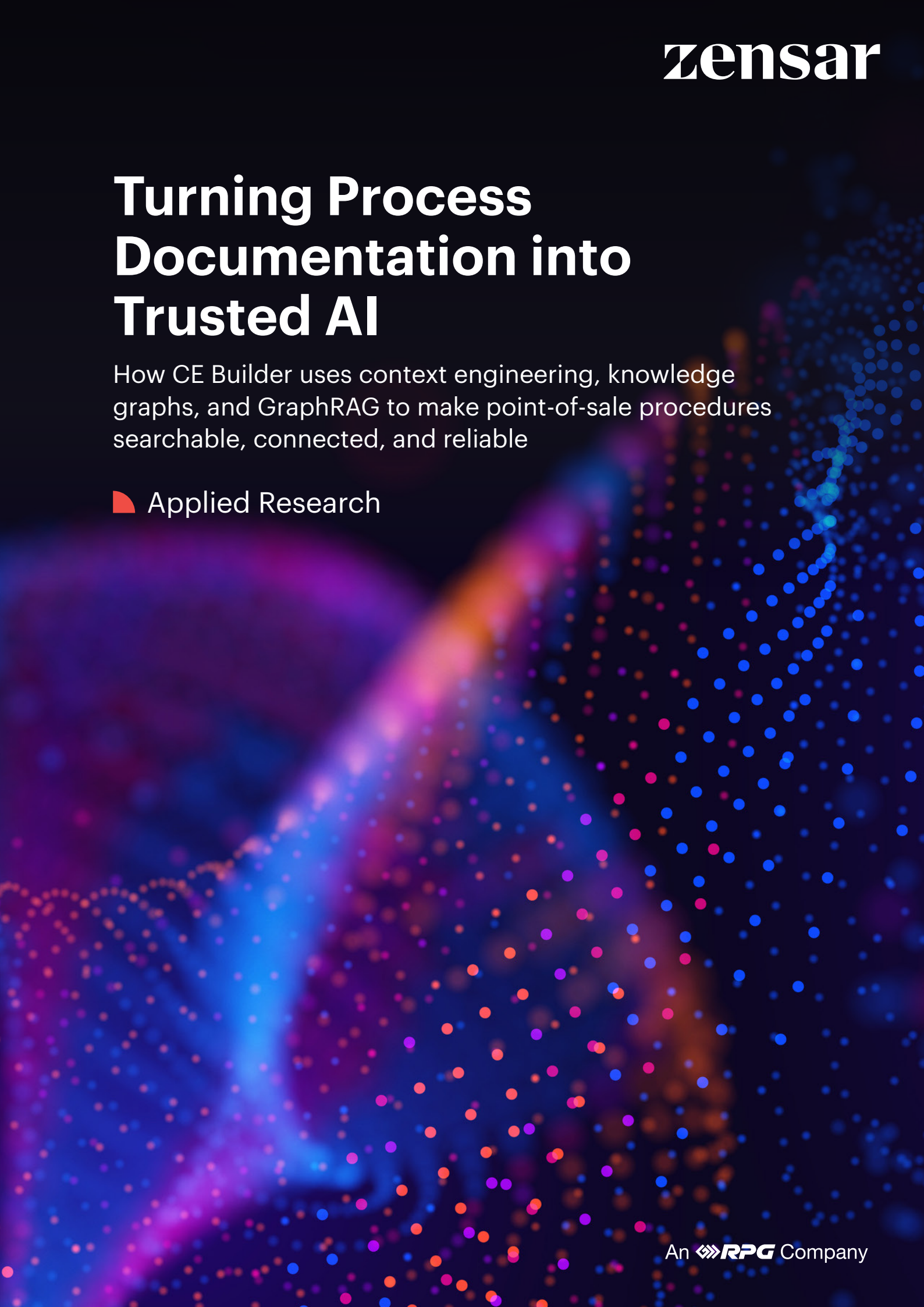
zensar

Turning Process Documentation into Trusted AI

How CE Builder uses context engineering, knowledge graphs, and GraphRAG to make point-of-sale procedures searchable, connected, and reliable

Applied Research



An RPG Company

Executive Summary

Retail operations run on procedure. Every sale, return, exchange, and exception follows a defined sequence of steps, and those sequences are documented in training guides, business requirements documents, and quick-reference sheets. In practice, that knowledge is hard to use. It sits in hundreds of PDF and Word files, much of it drawn as flowcharts and screen mockups that conventional search cannot read.

CE Builder is a platform that addresses this problem directly. It reads point-of-sale (PoS) documentation with a vision model, converts the extracted steps into a structured dataset, and loads that dataset into a Neo4j knowledge graph in which every step is linked to the step that follows it and to the step that handles its failure. A GraphRAG copilot then lets users ask questions in plain language and receive answers drawn only from the relevant part of the graph, together with a confidence score and the path used to reach the answer.

The project is built on one principle, context engineering: the quality of an AI answer depends less on how a question is worded than on what information the model is shown. CE Builder controls that information at every stage by selecting it, structuring it, limiting it, and verifying the result.

In its current form, CE Builder has consolidated 337 source documents, more than 6,500 pages in total, into a graph of 71,836 structured process steps. It runs entirely on local infrastructure with no paid API usage, so internal documentation never leaves the organization's environment. This paper describes the problem, the approach, the architecture, the governance controls, and the business value of the platform, and outlines where it can go next.

1. The Business Problem

Knowledge that exists but cannot be used

PoS procedures are among the most heavily documented processes in retail. They have to be, because the cost of getting a transaction wrong is immediate and visible to the customer. The documentation behind this project illustrates the scale of the challenge.

Source material	Volume	Characteristics
Business Requirements Documents (BRDs)	13	500+ pages each, formal specifications, and flowcharts
Training tips	291	1–2 pages each, quick-reference format
Help guides	33	1–2 pages each, task-focused
Total	**337 documents**	**6,500+ pages**

This is not a shortage of information. It is a surplus of information in a form that people cannot easily use. Three specific failures stand out.

Search cannot read the content that matters most. The logic of a PoS process is usually shown visually: a flowchart that branches on success and failure, or a screen mockup that shows which button leads where. Keyword search works on text, so it finds the document that mentions a topic but not the step the user needs.

Documents describe processes, but people ask about situations. A store associate does not ask for chapter four of a requirements document. They ask what happens after a card is declined, or which screen follows a barcode scan. The answer is a chain of steps that may span several pages, or several documents, and nothing in the source material is organized around that kind of question.

General-purpose AI is not a safe shortcut. A large language model can answer fluently about almost anything, which is exactly the risk. Without grounding in the actual documentation, it can invent steps that do not exist, quote figures that were never in the source, or expose sensitive details. Sending internal procedure documents to an external AI service also raises privacy questions and adds a cost per query.

What this means for the business

When process knowledge is hard to retrieve, organizations typically see the same pattern: longer onboarding for new staff, inconsistent execution from one location to another, frequent escalation of routine questions to experienced colleagues, and a growing reliance on individual memory rather than documented procedure. Training material also becomes harder to maintain, because nobody can easily see how the pieces fit together or where they contradict each other.

The goal of CE Builder is to remove that friction at the source, by turning documents into a structure that both people and AI systems can use reliably.

2. The Approach: Context Engineering

Beyond prompt engineering

Much of the early work with language models focused on prompt engineering, the craft of wording an instruction so that a model responds well. Prompt wording matters, but it has a hard ceiling. If the model is not given the right facts, no phrasing will make it produce them reliably. Context engineering takes the opposite view. It treats the information supplied to the model as the main design problem: what the model sees, in what form, and how much of it. CE Builder is named for this discipline, and its architecture applies it through four practices.

Practice	What it means	Where CE Builder applies it
Select	Find the right information for the question	Vector search and fuzzy keyword matching locate the relevant entry steps
Structure	Organize information so its meaning is explicit	Each step becomes a graph node with its actor, inputs, and screen, linked by typed relationships
Limit	Show the model only what it needs	Graph traversal expands a small neighborhood around the entry steps rather than loading the whole dataset
Verify	Check the output before it reaches the user	Five hallucination guards validate every answer

Why a graph
Process knowledge is relational by nature. A step has meaning because of what comes before it, what follows it, and what happens when it fails. Tables and text chunks capture this poorly, because the relationships are implied rather than stored.

A knowledge graph makes the relationships explicit. In CE Builder, each process step is a node carrying its actor, description, required inputs, and screen. The links between nodes, such as NEXT_STEP and ON_FAILURE, record how the process actually flows. A question like "what happens if this fails?" then becomes a simple traversal of the graph instead of a search through paragraphs.

This structure also supports the "limit" practice. Because the graph knows which steps are connected, the system can hand the language model a small, coherent slice of a process of 71,836 steps and be confident that the slice contains what the question needs.

3. Solution Overview

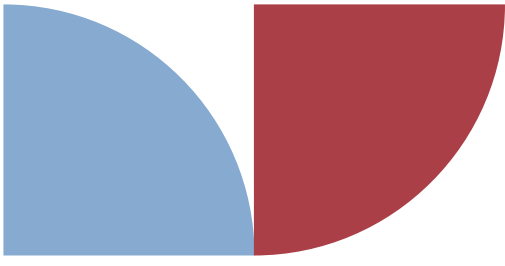
CE Builder converts documents into answers through five stages.

- **Read.** PDF pages are rendered as images, and a vision model reads the flowcharts, screens, and instructions, extracting each step and its connections into a structured file of 71,836 rows.
- **Structure.** Those rows, along with any unstructured documents, are converted into graph nodes and edges, and step names are turned into semantic embeddings.
- **Store.** The graph is loaded into Neo4j and the embeddings into a vector store.
- **Retrieve.** A user's question is matched to the most relevant steps by meaning, and the surrounding part of the process is expanded from those starting points.
- **Answer.** A local language model answers using only that retrieved context, and the answer is checked by the guardrails before it is returned.

Consider a user who asks, "What happens after scanning an item at the register?" The copilot matches the question to the scan step in the vector store, even if the wording differs from the documents. It follows the graph outward to the connected steps, including any failure paths, and passes that subgraph to the language model. The user receives a plain-language answer along with a confidence score, the graph path that supported it, and an interactive view of the subgraph.
That last element matters for business adoption. The answer is not a black box. Anyone can see which steps it was built from and judge whether the result is credible.

4. Architecture

CE Builder is organized as seven modules, each with a single responsibility. The backend is written in Python and exposed through FastAPI, and the user interface is built with React and Vite. A pipeline orchestrator runs the ingestion and loading stages in sequence, while the visual extraction step runs separately as preprocessing.



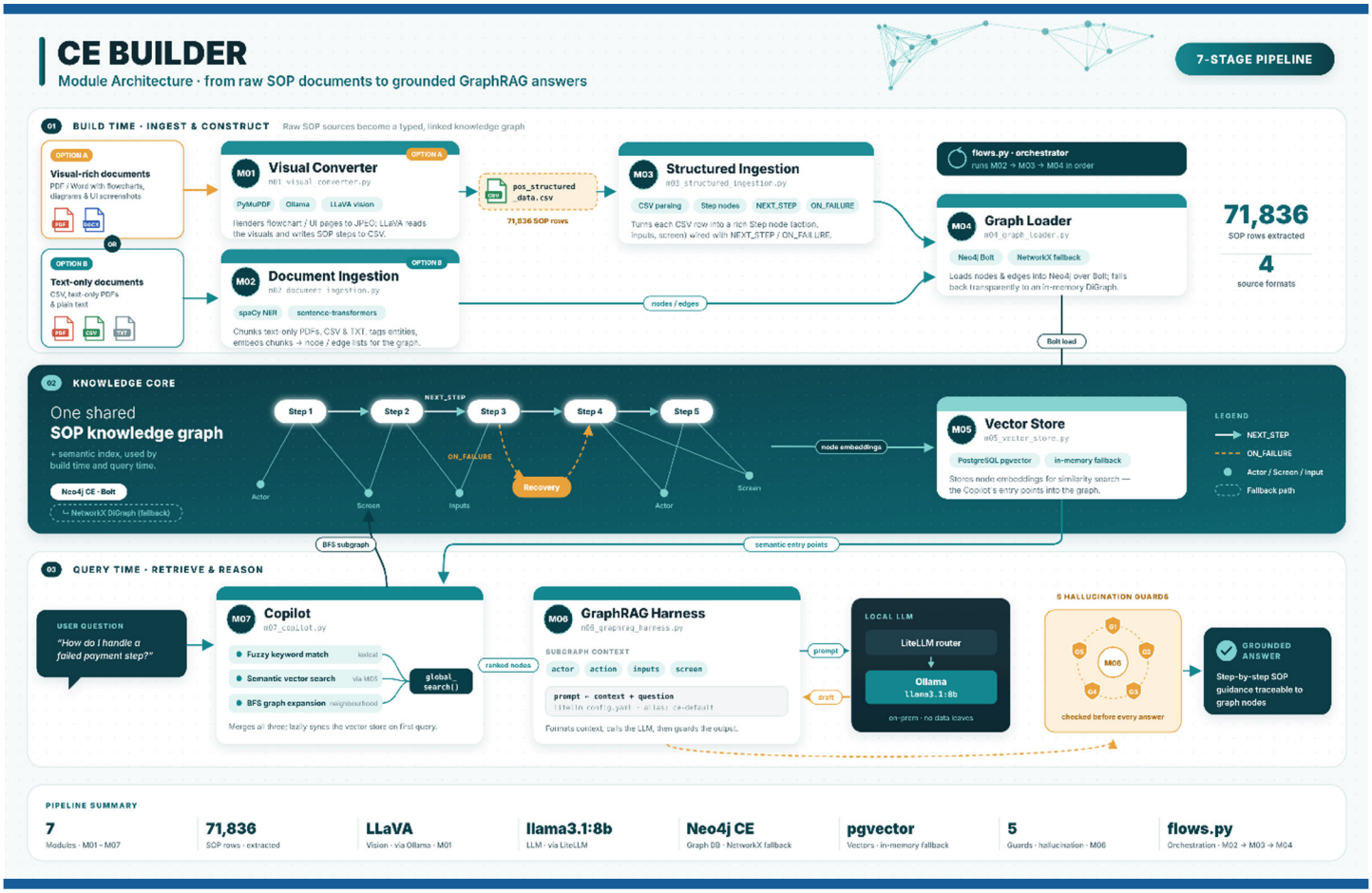


Figure 1: CE Builder Module Architecture

BFS subgraph

When the Copilot (M07) receives a question, it first finds a few entry nodes in the knowledge graph using fuzzy keyword matching and semantic vector search (M05). From those entry nodes, it runs a breadth-first search (BFS) outward, collecting the nodes directly connected to them, then their neighbors, up to a set number of hops. This pulls in the preceding and following steps along NEXT_STEP edges, the recovery paths along ON_FAILURE edges, and each step's actor, screen, and input nodes. The result is a small, focused subgraph that holds the full procedural context around the question.

Figure description

Figure 1. CE Builder module architecture. The pipeline has three layers. ***Build time (top):*** Source documents go one of two ways depending on their content. Option A sends visually rich PDF and Word files (flowcharts, diagrams, UI screenshots) to M01 Visual Converter, where the LLaVA vision model extracts SOP steps into a structured CSV that M03 Structured Ingestion turns into Step nodes linked by NEXT_STEP and ON_FAILURE edges. Option B sends text-only PDFs, CSV, and plain-text files to M02 Document Ingestion for entity extraction and embedding. Both routes feed M04 Graph Loader, under the control of flows.py. ***Knowledge core (middle):*** a shared SOP knowledge graph stored in Neo4j (with an in-memory NetworkX fallback), alongside node embeddings in M05 Vector Store (pgvector). ***Query time (bottom):*** M07 Copilot combines fuzzy keyword matching, semantic vector search, and BFS subgraph expansion. It passes the ranked context to M06 GraphRAG Harness, which prompts a local llama3.1:8b model through LiteLLM and Ollama. The draft answer passes five hallucination guards before it is returned as grounded, step-by-step SOP guidance.

M01: Visual Converter
The entry point for visual documentation. The module uses PyMuPDF to render every PDF page as an image and sends each image to LLaVA, a vision-language model running locally through Ollama. The model reads the flowcharts, screen mockups, and step-by-step instructions and returns the process logic as structured connections — for example, a login page leading to a dashboard on success. Pages are processed in parallel threads to keep extraction time manageable. The output is a single CSV file in what the project calls the Golden Schema, which holds all 71,836 extracted steps.

M02: Document Ingestion
The counterpart for unstructured material. This module reads PDF, CSV, Markdown, and plain text files, splits the content into chunks, extracts named entities with spaCy, and generates embeddings with sentence-transformers. It outputs node and edge lists in the same format the rest of the pipeline expects, so unstructured content and extracted visual content end up in one graph.

M03: Structured Ingestion
The module reads the Golden Schema CSV and converts each row into a rich Step node carrying its actor, description, required inputs, and screen. It also creates the edges that describe the process flow, chiefly NEXT_STEP and ON_FAILURE. This is where a flat file becomes a connected structure.

M04: Graph Loader
The loader writes nodes and edges into Neo4j Community Edition through the Bolt driver. If Neo4j is not available, it falls back transparently to an in-memory NetworkX directed graph, so the system still runs with no additional setup. That fallback is valuable for demonstrations and development, and it means a missing database never blocks the rest of the pipeline.

M05: Vector Store
The vector store holds the embeddings of the step names and supports similarity search. It uses PostgreSQL with the pgvector extension and falls back to an in-memory structure when a database is not present. Its job is to turn a user's question into a set of entry points in the graph, even when the question does not share words with the documentation. A question about "scanning a barcode" can reach a step called "Scan Item" because their embeddings are close.

M06: GraphRAG Harness
The harness turns a retrieved subgraph into an answer. It formats each step's actor, action, inputs, and screen into the prompt, then calls the language model through LiteLLM, which routes to a local Ollama model (llama3.1:8b) under a single configured alias. Before anything is returned, the answer passes through five hallucination guards, described in the next section.

M07: Copilot
The copilot is the entry point users interact with. It combines fuzzy keyword matching, semantic vector search, and breadth-first graph expansion in a single global search. On the first query, it synchronizes the vector store with the graph, then it finds entry steps, expands the surrounding neighborhood, and hands the result to the harness. It returns the answer together with confidence scores and the graph path used.

5. Trust, Governance, and Privacy

Five guards between the model and the user
A fluent but wrong answer is worse than no answer in an operational setting, so CE Builder treats verification as a core feature, not an afterthought. The harness applies five checks to every response.

1. **Grounding.** The answer must be supported by the retrieved subgraph, not by the model's general training.

2. **No numeric fabrication.** Figures and quantities that do not appear in the retrieved context are rejected.
3. **Relationship validity.** The answer may not assert connections between steps that do not exist in the graph.
4. **Confidence floor.** Low-confidence answers are flagged instead of presented as fact.
5. **PII redaction.** Sensitive personal information is removed before the response reaches the user.

These checks do not make the underlying model smarter. They make its failures visible and contained, which is what a business deployment requires.

Local-first by design

Every component in the platform can run on a single local machine. The vision model, language model, embedding model, graph database, and vector store all operate without calling an external service. This gives the organization three concrete advantages.

- **Data stays in the environment.** Internal procedure documents are never transmitted to a third party.
- **No per-query cost.** There is no API metering, so usage can grow without a matching increase in spend.
- **Predictable operation.** The system does not depend on an external provider's availability or pricing.

The fallbacks built into the loader and the vector store reinforce this. The platform degrades gracefully instead of failing when an optional service is absent.

6. Business Value

The value of CE Builder comes from changing what documentation is: from a static archive to be searched, into a structure that can answer questions.

Faster, more direct answers. A user asks a question in their own words and receives the relevant steps, instead of reading through several long documents to assemble the answer manually.

More consistent execution. Every answer is built from the same documented source and shows its supporting path. Two people asking the same question get the same grounded answer, which reduces variation between locations and shifts.

Faster onboarding. New staff can ask the copilot directly instead of relying on a colleague's memory. Experienced employees are interrupted less often with routine questions.

Reduced dependence on tribal knowledge. Procedures previously understood only by long-tenured staff become accessible because the knowledge is captured in the graph, not held in individual memory.

Trustworthy AI adoption. Many organizations want to use language models but cannot accept uncontrolled answers. By restricting the model to retrieved context and verifying the output, CE Builder offers a path to using AI in an operational setting with visible evidence for every answer.

Low and predictable cost. The open-source, local stack removes licensing and per-query charges. The cost of the platform is the cost of the hardware it runs on.

A reusable pattern. Nothing in the pipeline is specific to PoS content beyond the source documents and the schema. The same approach applies to any process-heavy documentation, which makes the investment portable.

7. Applications

The immediate use case is operational support for PoS procedures, but the same capability extends to several related scenarios.

- **Frontline assistance.** Store staff ask how to complete or recover a transaction and receive the steps for their situation.
- **Training and onboarding.** Learners explore a process interactively, following the graph from one step to the next, rather than reading it linearly.
- **Support desks.** Support teams trace a customer's issue through the documented flow and see where a failure path leads.
- **Documentation review.** Because every process is now a connected structure, gaps become visible, such as steps with no defined next step or failure path. This gives documentation owners a way to find and correct problems that are hard to see in prose.
- **Other domains.** Any organization with procedures stored as documents and diagrams, such as claims handling, onboarding, compliance workflows, or technical runbooks, can apply the same pipeline with a different schema.

8. Considerations and Limitations

A balanced assessment should be clear about what the current platform does and does not do.

Model capability: The language model runs locally at the 8-billion-parameter scale. That choice delivers privacy and zero API cost, but it limits reasoning depth compared with the largest hosted models. The guardrails are designed to contain this limitation, and the model sits behind a configuration alias, so a larger model can be substituted without changing the pipeline.

Extraction quality depends on source quality. The vision model reads what the images show. Low-resolution scans, dense diagrams, or unusual layouts can reduce extraction quality, and a review step for critical processes is sensible.

Database scale. Neo4j Community Edition is well suited to this workload, but does not offer clustering. A high-availability production deployment at larger scale would require a different edition or a managed graph service.

Processing time. End-to-end processing on a local machine takes time, particularly the vision extraction of hundreds of document pages. The system is therefore well suited to batch processing of documentation, with results served from the prepared graph.

Document currency. The graph reflects the documents it was built from. When procedures change, the affected documents need to be re-ingested so the graph stays current.

9. Roadmap

Several directions would extend the platform without changing its core design.

- **Incremental updates,** so that changed documents refresh only the affected part of the graph.
- **Model flexibility,** allowing larger local or hosted models to be selected per deployment through the existing configuration layer.
- **Broader inputs,** extending ingestion to additional file formats and image sources.
- **Process analytics,** using graph algorithms to examine flows for dead ends, loops, and unusually complex paths.
- **Additional domains,** with schema templates for other process-heavy areas of the business.

10. Conclusion

Organizations rarely lack documentation. They lack a way to use it. CE Builder addresses that gap by applying context engineering from end to end: it extracts process logic from the visual material that search cannot read, structures it as a graph in which relationships are explicit, limits what the language model sees to the relevant slice, and verifies every answer before it is shown.

The result is an AI copilot that behaves less like a confident guesser and more like a careful colleague who can point to the page it learned from. With 337 documents consolidated into a graph of 71,836 steps, a fully local deployment, and no per-query cost, CE Builder demonstrates that reliable AI for operational knowledge does not require sending sensitive documents outside the organization. It requires putting the right context in front of the model.

References

Anthropic. (2025, September 29). Effective context engineering for AI agents. Anthropic Engineering Blog.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy 2008) (pp. 11–15).

He, X., Tian, Y., Sun, Y., Chawla, N. V., Laurent, T., LeCun, Y., Bresson, X., & Hooi, B. (2024). G-Retriever: Retrieval-augmented generation for textual graph understanding and question answering. In Advances in Neural Information Processing Systems 37 (NeurIPS 2024).

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12), Article 248, 1–38.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In Advances in Neural Information Processing Systems 33 (NeurIPS 2020) (pp. 9459–9474). Curran Associates.

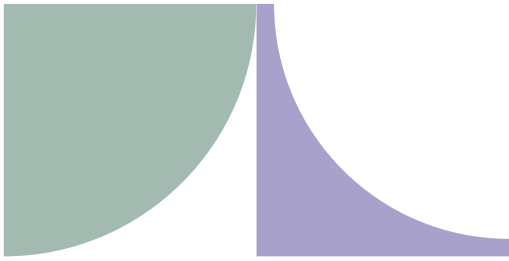
Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2023). Visual instruction tuning. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023) (pp. 34892–34916).

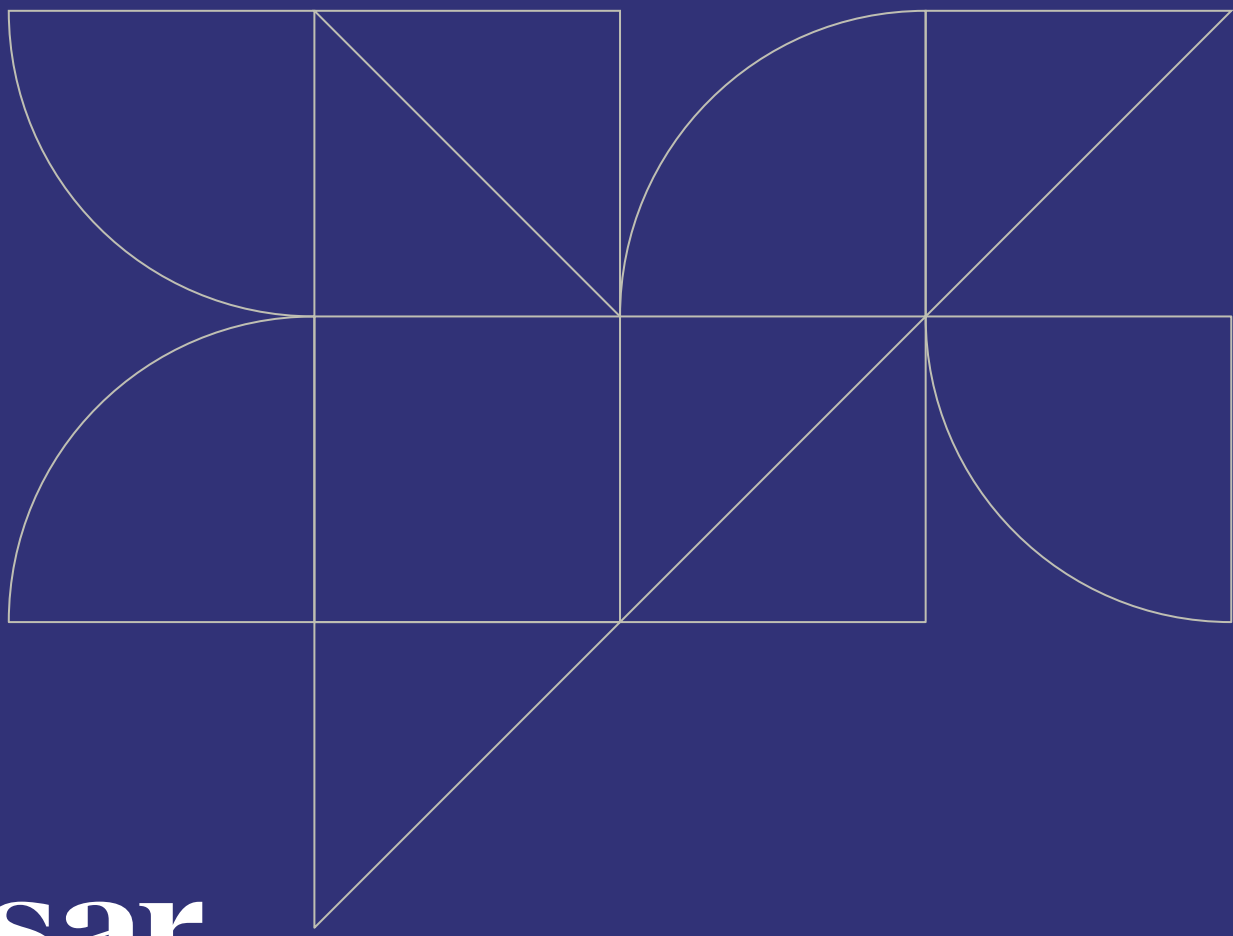
Neo4j, Inc. (n.d.). Neo4j operations manual: Introduction (Community and Enterprise Editions). Retrieved October 17, 2026, from

Ollama. (n.d.). Ollama: Get up and running with large language models [Software]. Retrieved October 17, 2026, from

pgvector. (n.d.). pgvector: Open-source vector similarity search for Postgres [Software]. Retrieved October 17, 2026, from

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Association for Computational Linguistics.





zensar

An RPG Company

At Zensar, we're 'experience-led everything.' We are committed to conceptualizing, designing, engineering, marketing, and managing digital solutions and experiences for over 145 leading enterprises. Using our 3Es of experience, engineering, and engagement, we harness the power of technology, creativity, and insight to deliver impact.

Part of the $4.8 billion RPG Group, we are headquartered in Pune, India. Our 10,000+ employees work across 30+ locations worldwide, including Milpitas, Seattle, Princeton, Cape Town, London, Zurich, Singapore, and Mexico City.

For more information, please contact: **info@zensar.com | www.zensar.com**