

Knowledge-Graph-Governed AI for Engineering Design: A Case Study in S-Lay Inline Structures

Authors: Sreekanth Manakkattil Sivaraman; Jagannatha Venkataramana Reddy

Indian Institute of Technology Bombay

email: sreekanth.sivaraman@gmail.com

Indian Institute of Technology Madras

email: jagannathavenkataramanareddy1@gmail.com

Abstract

Engineering organizations store reusable design knowledge in drawings, reports, finite-element studies, spreadsheets, review comments, and expert memory. Much of it is difficult to reuse because component identity, assembly topology, load case, assumptions, response location, and design reasoning are not represented as connected records. This paper presents a case study in applying state-of-the-art AI methods to industrial structural design, using subsea inline structures installed by S-lay as the engineering domain. The work is positioned as a continuation of two earlier S-lay inline-structure studies by the authors: those studies generated the engineering evidence base, and the present paper examines how that evidence can be converted into an ontology-grounded, reviewable AI4D workflow for professional engineers. The proposed method is described independent of any particular software repository. It separates component definitions, assembly rules, behavior evidence, current problem representation, deterministic tools, and governed feedback into EDES, EDAS, EDIKB, EDPR, toolchain, and KEL layers. A large language model is used as a constrained language interface; deterministic tools validate, retrieve, build layouts, plot, and report. The paper explains the S-lay inline-structure ontology, FBS-OAM assembly framing, P-map/APF problem representation, hybrid RAG pipeline, and KEL feedback loop. It does not claim final design sizing, code compliance, validated optimization, or replacement of project-specific analysis.

Keywords: engineering knowledge management; AI4D; knowledge graph; hybrid RAG; S-lay installation; inline structure; FBS-OAM; P-map; APF; KEL

1. Introduction

This paper is the third step in a continuing research chain on S-lay inline structures. The first two studies [10], [11] developed the domain evidence base for inline-structure concepts and installation response. They also presented a taxonomy for typical components and structural assemblies and proposed parametric forms for the structural assemblies. Refer Table 1 for summary of the past, current and planned future work. Refer Figure 1 for a representative figure of the parametrization done in previous works.

The present work asks a different question: how can that engineering knowledge be organized so that modern AI methods can help a professional engineer retrieve, reason about, and improve it without hiding the assumptions or bypassing engineering review? Industrial structural design rarely starts from a blank sheet. Engineers reuse previous layouts, installation lessons, analysis reports, review comments, and local practices. For subsea inline structures, that reuse is difficult because design behavior depends on coupled geometry, contact, stiffness, mass, connector behavior, installation boundary conditions, and response location.

Step	Capability	Objective	Scope
1	Component Taxonomy	Identify the fundamental components of the assembly and classify them with ID tags for consistent identification	Ref [10]
2	Design Intuition	Characterise strain amplification mechanisms for each IW/EA component type as a function of geometry, length, elevation offset and attachment stiffness. Components can be classified into two groups as below	Ref [10]
		Type A1, B1 and C1: Applicable for IW-P, IW-A, EA-O	
		Type A2, B2 and C2: Applicable for EA-SB, EA-ST, IW-B	Ref [11]
3	Intuitive Assembly	Predict dominant mechanical behaviour(s) type (A, B, or C) for a given assembly configuration before analysis. Translate component-level intuition into assembly-level design judgement.	This Work
4	ML with reduced order FEA Models	Develop reduced-order models for rapid parametric sensitivity screening without full Abaqus execution. Increase the knowledge base by implementing Machine learning through surrogate models.	Future Work
5	Design Code Assessment	Automate DNV-ST-F101 and ASME VIII capacity checks against computed demands. Integrate code compliance as a primary concept filter.	Future Work
6	Iterative Analysis	Closed-loop automated parametric workflow combining simplified models, code checks, and structured result interpretation with minimal manual intervention.	Future Work
7	Configuration Optimization	Identify governing parameters and systematically adjust toward minimum peak strain configurations within applicable code limits.	Future Work

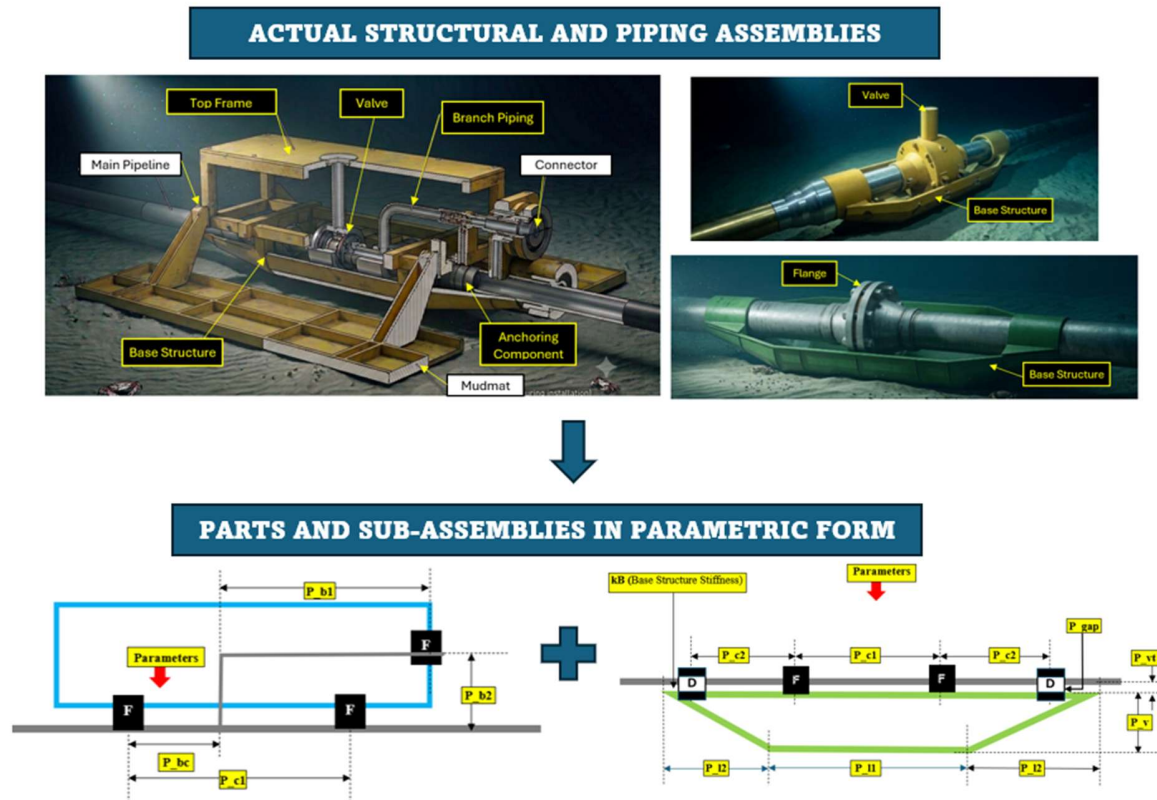


Figure 1: Parametric presentation of S-Lay Inline structures proposed in [10] and [11]

LLMs make this knowledge easier to query, but fluent text is not engineering validity. An LLM can produce a plausible layout while omitting a required support, applying evidence outside its domain, or drawing a concept that hides a missing association. The practical question is therefore not whether an LLM can answer; it is whether the answer is traceable, evidence-bounded, tool-checkable, and reviewable.

The case study treats engineering AI first as knowledge management and governance. The contribution is a layered method that connects an S-lay inline-structure ontology, explicit problem representation, governed retrieval, deterministic layout/plot tools, and a feedback loop for reviewed knowledge evolution. The repository implementation is used as a reproducible vehicle for the case study, while the research contribution is the engineering workflow and knowledge-governance approach that can grow as further evidence, correlation data, and ML models are added. Refer Figure 2 for the proposed workflow.

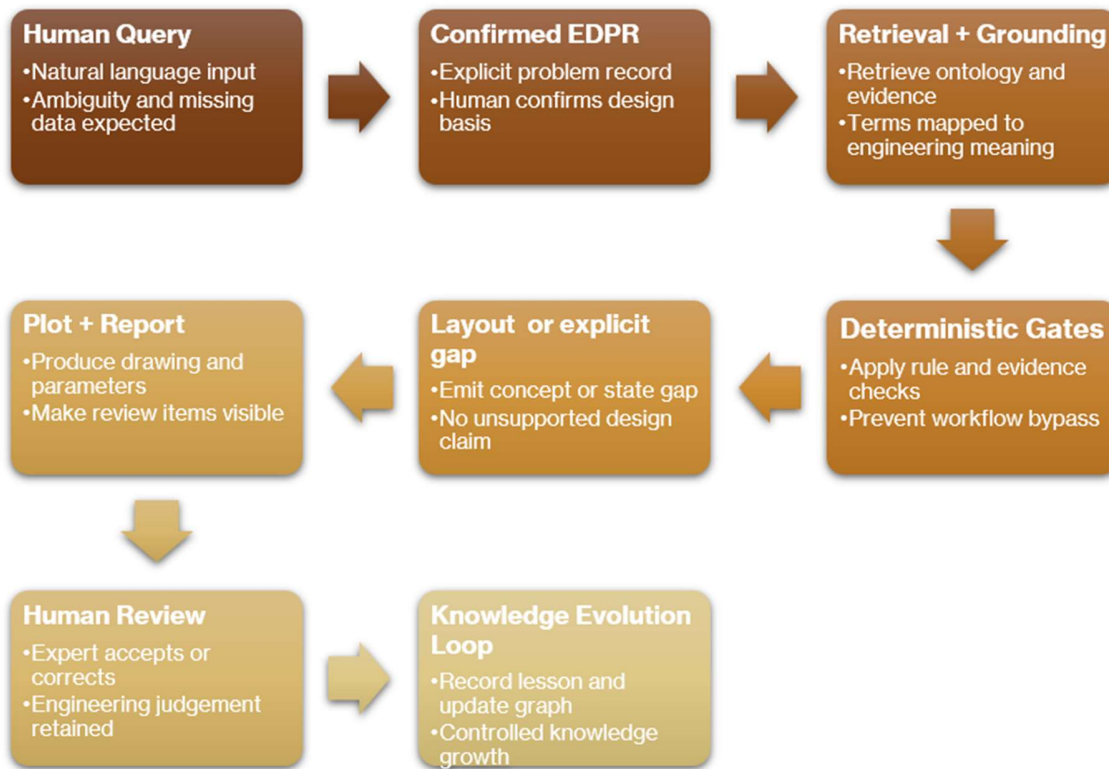


Figure 2 : Workflow Diagram

The intended reader is a practicing engineer who needs to understand how AI methods are being applied to a familiar structural-design problem. The paper therefore explains the toolchain and workflow as a transferable method. Any repository reference is secondary and optional, serving only to document one implementation of the method.

2. Foundations and Related Work

The case study combines four bodies of prior work. First, engineering-design AI has long used explicit knowledge representations to separate what a system is, what it does, and why it is needed. Gero's design-prototype work established design knowledge as reusable structure rather than isolated calculation notes [1]. Goel, Rugaber, and Vattam formalized structure-behavior-function modeling for complex engineered systems, giving a useful basis for separating physical objects, behavior, and purpose [2]. For S-lay inline structures, that separation is necessary but incomplete unless assembly features and relations are also represented.

Second, assembly representation research provides the language needed to describe connected engineered objects. The NIST Open Assembly Model work represents parts, assembly features, liaisons, positions, orientations, and tolerances [3]. This paper adopts that assembly view for inline-structure concepts: a valve, shroud, branch, base structure, top structure, and connector are not only shapes; they are associated objects with interfaces, positions, orientations, load-transfer roles, and contact responsibilities.

Third, design-problem formulation research motivates the EDPR layer. Dinar et al.'s Problem Map framework represents requirements, functions, artifacts, behaviors, and issues as linked problem-formulation records [4]. Automated Problem Formulation work similarly separates the action, product, and function of a request so that an AI system can distinguish designing, validating, comparing, ranking, clarifying, and explaining [5]. In this manuscript, EDPR applies those ideas to a bounded engineering workflow before any layout is generated, so the AI-assisted response starts from a reviewable design basis rather than from an uninspected prompt interpretation.

Fourth, recent AI systems motivate retrieval and tool governance, but they do not remove the need for engineering controls. Retrieval-augmented generation showed that language models can combine generated answers with explicit retrieved memory [6]. ReAct and Toolformer showed that language models can coordinate reasoning with external actions or tools [7], [8]. Human-in-the-loop LLM and knowledge-graph design frameworks further support the need for reviewable interaction between human experts, graph records, and generated concepts [9]. This case study uses these ideas conservatively: retrieval supports recall, tools enforce deterministic checks, and KEL records expert-reviewed corrections rather than treating raw model output as design authority.

The gap addressed here is therefore practical integration for professional engineering practice. The paper does not introduce a new general FBS, OAM, P-map, APF, or RAG theory. It shows how those ideas can be combined with prior S-lay inline-structure research to form a traceable AI4D case study where component identity, assembly topology, evidence applicability, visualization quality, and feedback governance must be visible to engineering reviewers. The planned next stage is to enrich the evidence base with additional simulation, correlation, and ML-derived knowledge rather than treating the current workflow as complete.

3. S-lay inline-structure ontology

Establishing an ontology is important because an AI-assisted design workflow must know what each engineering object means before it can reason about it. In an ILS layout, a valve, shroud, base structure, top structure, connector, branch pipe, and header pipe are not only drawn shapes; they carry functions, interfaces, constraints, load paths, and installation-response implications. Without an explicit ontology, an LLM may treat nearby objects as connected, confuse protection with support, or apply evidence to the wrong component.

This approach is consistent with established engineering information-model initiatives such as DEXPI, which develops open, vendor-neutral data models for process-industry engineering information exchange, especially P&IDs, BFDs, and PFDs. DEXPI's purpose is interoperability across tools and organizations through a common data model, including topology, symbols, attributes, and engineering data. The ontology proposed here is narrower: it is not a replacement for DEXPI or ISO 15926-based plant information models. It is a domain-specific ontology for S-lay inline-structure reasoning, intended to make component identity, assembly relationships, support/protection logic, and evidence applicability explicit for AI4D workflows.

The case-study knowledge model uses stable GD- identifiers so drawings, data records, plots, and validation reports refer to the same objects.

Code	Plain name
GD-HdPipe	Header pipe
GD-BrPipe	Branch pipe
GD-B	Branch assembly
GD-VLV	Valve
GD-TP / GD-TT	Thick / tapered thick pipe
GD-SH	Shroud
GD-ST	Top structure
GD-SB	Base structure
GD-Con	Connector/support

The source papers used related external-structure notation such as EA-ST and EA-SB. This manuscript uses unified case-study notation GD-ST and GD-SB; the change is terminology only. The underlying S-lay inline-structure evidence base comes from the two source ILT studies listed in the references [10], [11].

Associations are as important as components. A branch connector must terminate at a declared feature, not merely appear near a top frame. A base structure may own the roller-contact envelope while the pipeline or valve remains the structural section. These relations must be explicit because they cannot be inferred reliably from drawing proximity.

4. FBS-OAM for structural assembly knowledge

Function-Behavior-Structure (FBS) is useful because inline structures exist to perform functions, produce behavior, and use physical structure [1], [2]. For this domain, FBS alone is not enough because the design object is an assembly. The proposed toolchain therefore uses an FBS-OAM knowledge view: FBS captures function, behavior, and structure; OAM-style concepts make objects, assembly features, associations, positions, orientations, and load-transfer methods explicit [3].

FBS-OAM item	ILS interpretation
Function	Route flow, protect valve, support connector, permit installation, reduce strain
Behavior	Strain, moment, contact load, support reaction, stiffness transition
Structure/object	Pipe, valve, branch, shroud, GD-ST, GD-SB, connector, thick section
Assembly feature	Weld end, connector slot, support interface, branch terminal, contact area
Association	Branch-to-header, branch-to-GD-ST, support-to-pipe, valve-to-protection
Position/orientation	L branch, Z branch, horizontal/vertical connector, valve station, connector spacing

This matters because structural behavior is driven by interactions. A header valve and a branch valve trigger different support logic. A shroud around a stiff component is not only a cover; it changes contact and stiffness behavior. A connector system may reduce one local issue while increasing stiffness elsewhere.

5. P-map, and APF

P-map and APF are adopted because the ILT design request is not only a component-selection problem; it is also a problem-formulation and optimization problem. The P-map framework proposed by Dinar et al. [4] is useful because it records the current problem state as requirements, artifacts, behaviours, issues, and links. This fits the ILT workflow, where a query may include physical components, missing installation assumptions, target behaviours such as strain or bending moment, and unresolved design issues. The APF framework by Li et al. [5] complements this by separating the requested action, product, and function. This is important for ILT design because the same user request may require designing a layout, comparing alternatives, reducing strain, protecting a valve, validating a branch connection, or asking for missing inputs. Combining P-map and APF forces the workflow to identify what is being designed, why it is being designed, what behaviour is being optimized, what evidence is needed, and what assumptions remain unresolved. This reduces the chance that the LLM will respond from a partial reading of the query or produce a plausible but ungrounded design response.

Framework item	Example in an ILS request
APF action	design, compare, rank, validate, explain, clarify
APF product	ILT layout, valve protection concept, branch assembly
APF function	route flow, protect valve, support connector, reduce strain
P-map artifact	GD-B, GD-ST, GD-SB, GD-VLV, ILT-Z-*
P-map behavior	peak strain, bending moment, contact load, support stiffness
P-map issue	missing roller geometry, unresolved topology, weak evidence

The governed workflow requires EDPR confirmation before layout generation. In engineering terms, this is a design-basis check before calculation or drawing.

6. Transferable AI4D toolchain and knowledge architecture

The proposed AI4D toolchain separates knowledge by responsibility. The layers may be implemented in files, databases, graph stores, document stores, or enterprise engineering systems; the paper does not depend on a particular repository structure. In the present case study, simple human-readable files are useful because they make every engineering rule, evidence note, feedback record, and validation step auditable by a professional engineer. Refer Figure 3 for the summary of workflow and Table 2 for description of the layers.

Layer	Engineering responsibility	Typical implementation file types
EDES (Engineering Design Equipment Specification)	Component identity, geometry, parameters, interfaces, local constraints	JSON knowledge records; JSON schemas
EDAS (Engineering Design Assembly Specification)	Assembly topology, valid anchors, associations, contact ownership, section ownership	JSON assembly rules; JSON schemas; validator scripts
EDIKB (Engineering Design Intuition Knowledge Base)	Behaviour evidence, numeric response data, evidence limits, uncertainty, future-study candidates	CSV datasets; JSON graph records; Markdown evidence notes
EDPR (Engineering Design Problem Representation)	Current problem statement, objectives, constraints, unknowns, APF/P-map interpretation	Markdown prompts; JSON problem records; example EDPR files
Governed tools	Validation, retrieval, design gates, layout materialization, plotting, reporting	Python scripts; YAML settings; generated JSON/CSV/PNG/PDF reports
KEL (Knowledge Evolution Loop)	Human feedback, root cause, expert review, graph-change request, implementation evidence, promotion state	Markdown records; JSON schemas; templates; lifecycle reports; Python utilities

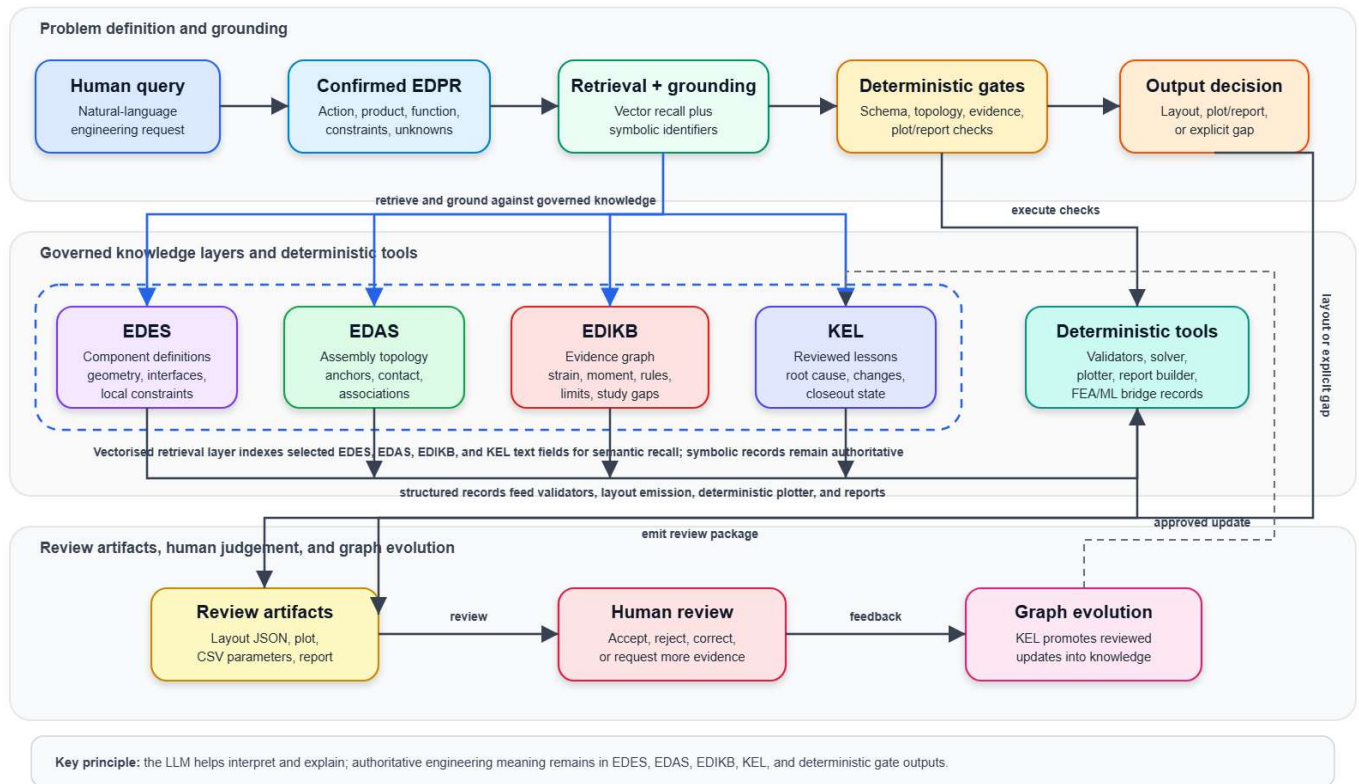


Figure 3: Nested toolchain architecture

A vectorized retrieval layer is implemented above EDES, EDAS, EDIKB and KEL to improve semantic recall while preserving the structured records as the authority. EDES, EDAS, EDIKB and KEL contain the reviewed component definitions, assembly rules, and behaviour evidence. The vectorised layer stores embeddings of selected record fields, descriptions, evidence notes, rule summaries, and KEL lessons so that natural-language engineering queries can retrieve relevant records even when the user does not use exact ontology terms. Retrieved candidates are then grounded back to symbolic identifiers such as GD-VLV, GD-SB, GD-ST, or GD-B and passed through deterministic gates. Therefore, vector search improves retrieval coverage, but it does not decide engineering validity.

This tool chain is intentionally conservative. It fails closed when a topology cannot be represented, when evidence is missing, or when a layout cannot be generated from repository-backed component and assembly definitions. In engineering practice, this behavior is more valuable than a fluent answer that hides missing assumptions.

6.1 EDES: component knowledge

EDES is the component-definition layer. It uses FBS-OAM Framework and records what each engineering object is before it is placed in an assembly. For S-lay inline structures, EDES records include the header pipe, branch pipe, valve, shroud, top structure, base structure, connector, thick pipe, and tapered pipe concepts. Each component record should state the component name, identifier, geometric parameters, interfacing regions of the component, contact limitations, and any local constraints. In a file-based implementation this is naturally represented as JSON because component attributes can be validated against a schema and reused by retrieval, plotting, and checking tools.

EDES content	Purpose	Example file type
Component identifier and plain name	Prevents the same object being described by multiple inconsistent names	JSON
Geometry and parameters	Provides dimensions and editable variables for layout and plotting	JSON
Interfaces and contact constraints	States what can connect, support, touch rollers, or require protection	JSON plus schema
Component validation rules	Checks whether required fields are present before use	Python validator plus JSON schema

6.2 EDAS: assembly and association knowledge

EDAS describes how valid components may be assembled using the OAM Framework. This layer is needed because engineering behaviour depends on relationships, not only individual parts. A branch line near a top structure is not automatically supported by it; the association must be explicit. A base structure may own roller contact while the pipe or valve remains the structural item being protected. EDAS therefore records topology, permitted anchors, required associations, support/protection rules, and contact ownership. JSON is suitable for the rule records, while Python validators are useful for checking whether a proposed layout satisfies the rules.

EDAS content	Purpose	Example file type
Assembly topology	Defines how to assemble various components to prepare a full layout	JSON
Required associations	Ensures branch-to-header, branch-to-GD-ST, valve-to-protection, and support-to-pipe links are explicit	JSON plus schema
Contact and section ownership	Distinguishes the object that rides rollers from the object being protected	JSON
Topology and association gates	Blocks layouts that omit compulsory supports, anchors, or protection	Python validator

6.3 EDIKB: behaviour evidence and engineering knowledge base

EDIKB is the evidence and behaviour layer of the workflow. It stores design knowledge as a graph linking source studies, evidence anchors, numerical data rows, components or assemblies, varied design parameters, response metrics, response regions, behaviour rules, design guidance, and evidence limits. In the current case-study implementation, EDIKB includes component strain behaviour, component bending-moment behaviour, topology-specific rules, derived features, correlation candidates, and future ML-study candidates. Its job is to connect a particular design query to the available knowledge on such designs, and the source table or figure from which the evidence was extracted. This structure lets retrieval find relevant behaviour evidence while deterministic gates check whether the evidence applies to the current EDPR. Gaps in the evidence are stored as future-study candidates for parametric FEA, correlation studies, surrogate models, or physics-informed ML.

EDIKB content	Purpose	Example file type
Numerical response dataset	Stores strain, moment, parameter, and case-study values	CSV
Knowledge graph records	Links claims to components, load cases, evidence limits, and sources	JSON
Source provenance	Tracks which paper, model, analysis, or review note supports a claim	JSON or Markdown
Correlation and ML-study candidates	Records gaps that need parametric FEA, surrogate models, or future ML work	Markdown or JSON

Representative EDIKB Evidence and Behaviour Graph

Design evidence is stored as linked source, component, parameter, response, rule, guidance, and future-study records

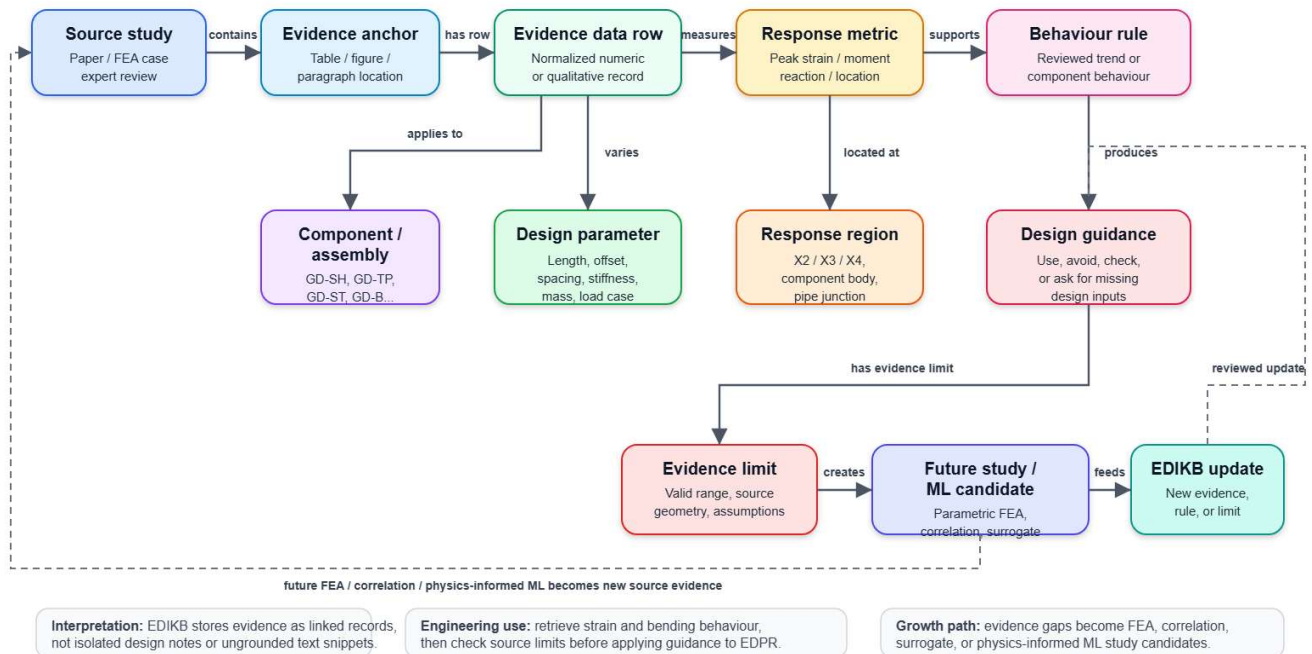


Figure 4: Representative illustration - EDIKB Evidence and Behavior Graph

6.4 EDPR: current problem representation

EDPR is the current-problem layer. It converts a natural-language request into an explicit problem record before the workflow generates a layout. EDPR should state the action, product, function, known inputs, missing inputs, assumptions, optimization target, required outputs, and planned retrieval. Markdown prompts are practical for describing how the LLM should parse a request. JSON records are practical for the resulting structured problem statement because they can be validated and passed to downstream tools.

EDPR content	Purpose	Example file type
Problem-understanding prompt	Guides the LLM to state the design basis before solution	Markdown
Structured problem record	Captures APF/P-map fields, assumptions, unknowns, and requested outputs	JSON
Example problems	Tests whether the workflow interprets recurring design requests consistently	JSON or Markdown
EDPR validation	Checks completeness before retrieval, layout, or plotting	Python validator plus JSON schema

6.5 Governed tools: validators, retrieval, plotting, and reporting

The tool layer is deterministic. It should not replace engineering judgement; it should enforce checks that should not depend on the wording of a prompt. Typical tools include validators for EDES, EDAS, and EDPR; a governed workflow runner; design-rule checks; plot generation; plot readability checks; and report packaging. Python is suitable for these tools because it can read JSON/CSV/YAML inputs, apply deterministic rules, and emit review artifacts. YAML is useful for plot style settings because it separates figure styling from engineering logic.

Tool function	Purpose	Example file type
Validators	Check schemas, required fields, topology, and associations	Python
Workflow runner	Executes EDPR confirmation, retrieval, gates, layout, plot, and report sequence	Python
Plotter and plot checker	Creates reviewable drawings and checks visibility/readability gates	Python plus YAML style file
Review artifacts	Gives the engineer the layout, parameters, assumptions, and gaps	JSON, CSV, PNG, PDF, Markdown

6.6 Deterministic plotter and downstream model emission

The plotter is an important governed tool because visual review is central to structural engineering. A text-only response can state that a valve is protected, a branch is supported, or a connector is attached to a top frame, but the engineer must be able to see whether the emitted layout actually shows that relationship. General image generation is not sufficient for this purpose because generated drawings can hallucinate geometry, omit small labels, invent connections, or visually imply associations that are not present in the engineering record. The plotter must therefore be deterministic: it should draw only from the structured layout data emitted from EDES and EDAS records.

In the proposed workflow, EDES and EDAS should be usable to emit an intermediate layout file. That file is not only a plotting input; it is a reusable engineering object. The same component identities, coordinates, dimensions, connection types, support associations, contact ownership, and assumptions can be used to create a readable concept plot, export a parameter table, and later seed finite-element model generation. This is important for the next stage of the framework, where additional FEA results, correlation studies, surrogate models, and physics-informed machine-learning workflows can be linked back to the same component and assembly identifiers.

Plotter input or output	Engineering purpose	Example file type
Structured layout record	Captures components, coordinates, dimensions, associations, connection types, and assumptions	JSON
Parameter export	Separates dimensions, weights, support spacing, and assumed values from the drawing	CSV or JSON
Reviewable plot	Shows topology, support/protection logic, connection labels, and critical locations	PNG or PDF
Plot validation report	Checks label visibility, required associations, missing outputs, and possible drawing ambiguity	JSON or Markdown
FEA/ML bridge record	Preserves the same layout identifiers for meshing, load-case setup, response mapping, and future physics-informed ML	JSON, CSV, or model-builder input

The plotter therefore acts as a controlled translation layer between AI-assisted reasoning and engineering review. It also prepares the workflow for future model automation: a layout that can be plotted deterministically from EDES/EDAS data can also be converted, with additional rules and verification, into an FEA pre-processing record. Once FEA results are generated, the response fields can be mapped back into EDIKB and later used for physics-informed ML or bounded surrogate models.

6.7 KEL: reviewed feedback and graph evolution

KEL is the controlled learning layer. It prevents ad hoc feedback from becoming official knowledge without review. A KEL cycle records the observed issue, grouped feedback, root cause, expert decision, proposed graph or tool change, implementation evidence, and closeout state. Markdown is useful for human-readable records; JSON schemas are useful for enforcing required fields; Python utilities are useful for validating records and promoting accepted changes. This is the layer that allows the workflow to improve after expert review while preserving traceability.

KEL content	Purpose	Example file type
Experience and feedback records	Captures what happened, what was wrong, and who reviewed it	Markdown plus JSON schema
Root-cause log	Separates missing knowledge, missed retrieval, weak gate, plot failure, or workflow bypass	Markdown
Graph-change request	States exactly what knowledge or rule should be changed	Markdown or JSON
Lifecycle and promotion utilities	Validates, promotes, and closes accepted lessons	Python scripts plus lifecycle reports

This separation supports failure diagnosis. A weak output may be caused by problem parsing, component definition, assembly representation, evidence retrieval, evidence applicability, tool execution, deterministic plotting, visualization, workflow bypass, or missing study data. KEL uses that diagnosis to improve the workflow without turning unreviewed comments into official knowledge. The same architecture can be implemented with different software choices as long as the gates, records, and human-review responsibilities are preserved.

7. Hybrid RAG pipeline

Section 4 introduced the vectorised layer as part of the toolchain architecture. In the Hybrid RAG pipeline, that layer provides semantic recall; the retrieved candidates are then grounded to EDES, EDAS, EDIKB, and KEL records before deterministic checks are applied.

Symbolic records are necessary for engineering authority, but exact keyword matching is not enough. Engineers may say "strongback" when the case-study ontology term is GD-ST, or "valve cannot ride rollers" when the grounded meaning includes GD-VLV, GD-SB, roller-contact limits, missing roller geometry, moment evidence, and prior KEL lessons.

Hybrid RAG is used as semantic recall, not as final authority [6]. The related tool-use literature supports the idea that language models can coordinate external actions, but in this workflow those actions remain gated by deterministic engineering checks [7], [8].

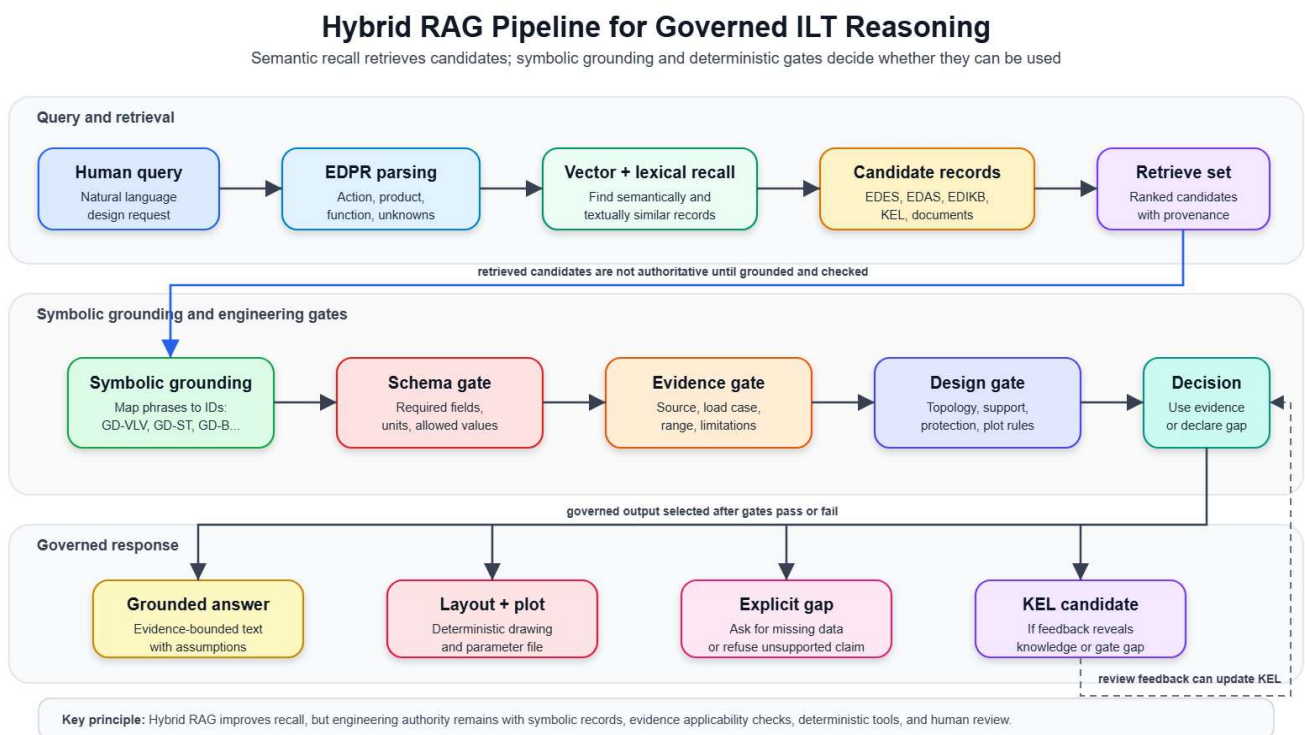


Figure 5: Hybrid RAG Pipeline implementation

Phrase	Candidate grounding
strongback / top frame	GD-ST, top-structure associations
base frame / roller protection	GD-SB, valve/base protection gates
valve cannot ride rollers	GD-VLV, GD-SB, missing clearance/load inputs
vertical connector	GD-B.variant = Z, ILT-Z-*, branch-to-GD-ST association
similar past mistake	KEL root-cause records and implemented gates
is this optimized?	EDIKB evidence, future-study/correlation-gap records

A semantically similar paragraph becomes useful only after grounding, evidence checking, and governance. The target is hybrid RAG for recall, symbolic grounding for engineering meaning, deterministic gates for validity, and KEL for loophole detection.

8. Governed workflow and KEL

The authoritative design path is a governed workflow, not an isolated prompt response. The workflow validates EDPR, retrieves context, solves or proposes layouts, materializes a reviewable concept, plots the concept, and packages review artifacts. A direct plot or lower-level tool output is not authoritative unless it passes through the required design-governance gates. This paper describes that workflow at method level; a repository or software implementation is only one possible way to execute it.

The workflow is deliberately stated as a sequence of responsibilities rather than as a sequence of software commands:

1. confirm EDPR problem understanding;
2. retrieve and ground EDES, EDAS, EDIKB, KEL, and relevant documents;
3. check topology, evidence, support/protection, and plot/report gates;
4. emit an EDAS-compatible layout, explicit gap, or request for missing information;
5. generate a reviewable plot and machine-readable report;
6. capture feedback through KEL when correction or learning is needed.

KEL records the experience, feedback, root cause, expert decision, implementation evidence, and promotion state. Its human-review role is consistent with recent human-in-the-loop LLM and knowledge-graph design work, but KEL adds lifecycle control for lessons and graph evolution [9]. Recent KEL work showed that root cause must be captured per implemented candidate. The same visible error can arise from missed retrieval, missing EDAS rule, insufficient EDIKB evidence, plot-report weakness, or workflow bypass.

Experience class	Principle captured
EDPR not shown before design	Confirm problem understanding before concept generation
Header valve without support/protection	Apply explicit GD-SB/protection logic where applicable
Branch connector outside GD-ST	Every GD-B branch needs declared GD-ST association
Evidence available but missed	Retrieve EDIKB/KEL evidence and limitations
Plot visibility failure	Treat plot readability/report completeness as gates
Missing correlation data	Create future-study candidates, not optimization claims

9. Where ML Fits

Machine learning should be introduced after the framework has defined variables, topology classes, evidence domains, and governance rules. ML is not a replacement for EDES, EDAS, EDIKB, EDPR, or KEL. It is a later layer that can operate on the structured data those layers create.

ML role	Use in the framework
Surrogate response prediction	Estimate strain, bending moment, contact load, or utilization for EDAS-valid candidates.
Similarity retrieval	Find historical or simulated cases closest to a new EDPR problem.
Gap detection	Identify sparse regions in topology, parameter, and evidence space.
Active learning	Select parametric FEA cases that most improve coverage.
Uncertainty estimation	Warn when recommendations leave known evidence domains.
Optimization	Rank EDAS-valid candidates after behavior prediction and constraint screening.

The overbend physics engine becomes important in this path. A validated physics program can generate structured simulation data across candidate regions of the design space. KEL can then record where the graph was insufficient, define future study candidates, and govern how new FEA/ML evidence enters EDIKB.

The principle is:

ML becomes useful only after the framework has defined what the variables mean, which assemblies are valid, what evidence exists, and where uncertainty remains.

This also changes how "design intuition development" should be described. The system should not claim intuition because the LLM produced a plausible recommendation. Instead, the framework accumulates structured experience: what was asked, what evidence was retrieved, what available correlation was missed, what gate failed, what root cause was found, and what future study is required. Over time, this creates a supervised pathway from human review and tool failures to more reliable rules, richer datasets, and eventually ML-assisted prediction.

10. Implementation Tests

This section presents two implementation tests conducted using GPT 5.5.

10.1 Implementation Test 1: ILT with branch valve with vertical connector

Step 1 User query

Provide an layout for following ILT:

- 16inch Pipeline
- 8inch Branch Pipe leading to Vertical Connector
- 8inch Valve on the Branch
- No Valves on the Header pipeline
- 90m Stinger Radius with 150MT pipeline tension

Provide info for

- Possible max strain on Pipeline
- Expected location of max strain

Step 2 LLM response (EDPR and query for human actions)

The LLM stated the human user's remaining actions before proceeding:

1. Confirm whether the branch line is a Z-branch ending in a vertical connector.
2. Confirm whether the 8 inch branch valve should be represented as an actual branch valve body, using estimated dimensions if exact dimensions are not available.
3. Confirm whether a top frame GD-ST is required around the branch valve and connector.
4. Confirm whether typical values may be used for missing details such as branch wall thickness, branch valve body size, connector body size, and component weights.
5. Confirm whether a layout plot is required after the concept proposal.

The LLM then emitted the problem understanding:

- The query describes a 16 inch header pipeline with an 8 inch branch pipe.
- The branch pipe leads to a vertical connector.
- The branch contains an 8 inch valve.
- There is no valve on the header pipeline.
- The installation basis is 90 m stinger radius and 150 MT pipeline tension.
- The user requested a layout proposal, possible maximum pipeline strain, and expected location of maximum strain.

The LLM mapped the problem to EDAS terms:

- A branch pipe leading to a vertical connector should normally use a GD-B Z-branch layout.
- A simple straight vertical branch is not sufficient.
- Because the valve is on the branch, not the header, a header base structure GD-SB is not automatically required.
- The branch valve and vertical connector should be contained within a top structure GD-ST unless the user confirms otherwise.
- For low-strain intent, the default connection system should avoid unjustified high-stiffness F2 or F2D.
- A P-S style support basis is preferred unless evidence justifies fixed supports.

The LLM proposed the likely concept after confirmation:

Item	Proposed representation
Header	GD-HdPipe, 16 inch pipeline
Branch assembly	GD-B, Z-branch variant
Branch valve	Represented on the branch horizontal leg, with valve body shown in plot
Support connection	Prefer S at branch/top-frame vertical connector unless fixed connection is justified
Header/top-frame connector system	Prefer low-strain P-S rather than defaulting to F2
Top frame	GD-ST, sized to contain branch tee, branch valve, and vertical connector end
Base structure	No GD-SB because there is no header valve or header component needing roller protection

The LLM gave a screening strain statement:

- Possible maximum pipeline strain: approximately 0.4% to 0.7%.
- This was stated as a screening estimate only, not an FEA result.
- Expected maximum strain location: header near the branch tee / local stiffness entry region.
- Secondary watch locations: near the GD-ST connector load-transfer points.

The LLM then asked the user to confirm the five human actions before producing the final layout proposal and plot.

Step 3 User response confirmation

1. confirmed to end with vertical connector
2. Use estimated dimensions
3. Yes Top frame required
4. Use typical
5. Layout plot required

Step 4 LLM response (layout generation and plot)

The LLM proceeded with the governed layout step, using typical and estimated dimensions. The generated concept used:

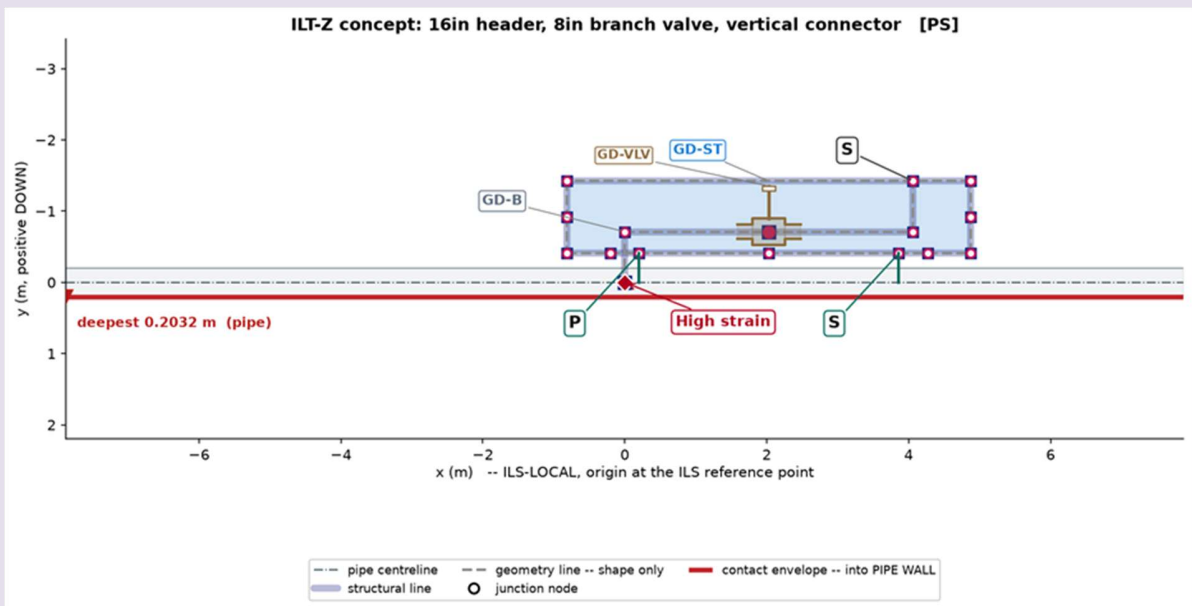
Design element	Selected representation	Basis
Header pipeline	GD-HdPipe	16 inch OD Base pipeline for local ILT assembly
Branch assembly	GD-B, Z variant	Vertical connector requires a Z-branch

Branch valve	Represented on the GD-B branch run using estimated dimensions and GD-VLV visual grammar	Valve is on the branch and must be visible for review
Top frame	GD-ST around branch tee, valve, and branch end	Required by user and by branch-valve containment rule
Connection system	PS intent	Avoid unjustified F2/F2D stiffness unless evidence supports it
Branch/top-frame tie	S connection from B.end to ST.top1	Vertical branch connector anchored to top frame using sliding support intent
Base structure	Not provided	No header valve or header component requiring roller-contact protection

The LLM generated a deterministic plot using the repository plotter.

The LLM reported:

Item	Statement
Possible max pipeline strain	Approximately 0.4% to 0.7%
Estimate status	Screening estimate only, not FEA
Expected primary location	Header near the branch tee / local stiffness entry region
Secondary watch locations	Near top-frame connector load-transfer points
Plot QA	Passed with automatic label corrections
High-strain marker	Included and anchored to B.tee
Remaining warning	Simplified connector representation; full GD-Con connector bodies and pipe-landing associations are not explicitly modelled



The LLM then asked for human review of:

1. Branch valve placement.
2. P / S label visibility and location.
3. High strain marker location.

Step 5 User response acceptance and recording instruction

1. Acceptable
2. Correct
3. Acceptable

Step 6 LLM response record creation

The LLM recorded the accepted case

10.2 Implementation Test 2: Inline valve

Step 1 User query

Design a layout for below Inline structure

- 20inch Pipeline
- 20inch Valve on the Headerline
- Valve cannot contact the stinger

Step 2 LLM response EDPR and remaining human actions

The LLM interpreted the request as an inline ILT layout with a 20 inch header pipeline and a 20 inch valve welded inline on the header line. Since the valve cannot contact the stinger, the valve should not ride rollers and should not take roller reaction directly.

The LLM asked the user to confirm:

- (1) whether the valve cannot ride or take reaction,
- (2) whether GD-SB base protection is acceptable,
- (3) whether typical valve dimensions may be used,
- (4) whether minimum added strain and bending moment is the target, and
- (5) whether a plot is required.

LLM emitted EDPR Problem Understanding

Item	Proposed representation
Header Pipeline	GD-HdPipe, 20 inch pipeline
Inline Component	20 inch valve welded inline on the header line
Valve contact condition	Valve cannot contact the stinger
Required design action	Protect the valve from roller contact
Likely protection component	GD-SB base structure
Main design target	Protect valve while minimizing added strain/bending on the pipeline

Step 3 User response confirmation

1. Cannot take reaction, nor ride
2. Yes
3. Use Typical
4. Yes, target minimal strain/moment
5. Yes, plot

Step 4 LLM response layout generation and plot

The LLM proposed an inline GD-VLV protected by a fitted GD-SB base structure. The base was sized to the valve envelope with limited clearance to avoid unnecessary elevation strain. A P-S support connection system was used. Each connector was represented as an explicit GD-Con landing on a short

GD-TP connector landing body rather than on the valve end node or plain header pipe. The deterministic plot showed the GD-SB, GD-TP landing bodies, GD-Con P-S connectors, and the protected valve layout.

Feature	Proposed arrangement
GD-VLV	Inline 20 inch valve centered on header
GD-SB	Base structure below valve, fitted to valve envelope
GD-TP	Short connector landing bodies on header outside valve envelope
GD-Con	Explicit connector elements from GD-SB to GD-TP
Connection system	P-S
Strain control	Avoid excessive GD-SB depth and large connector/elevation arm
Roller contact	Taken by GD-SB, not valve

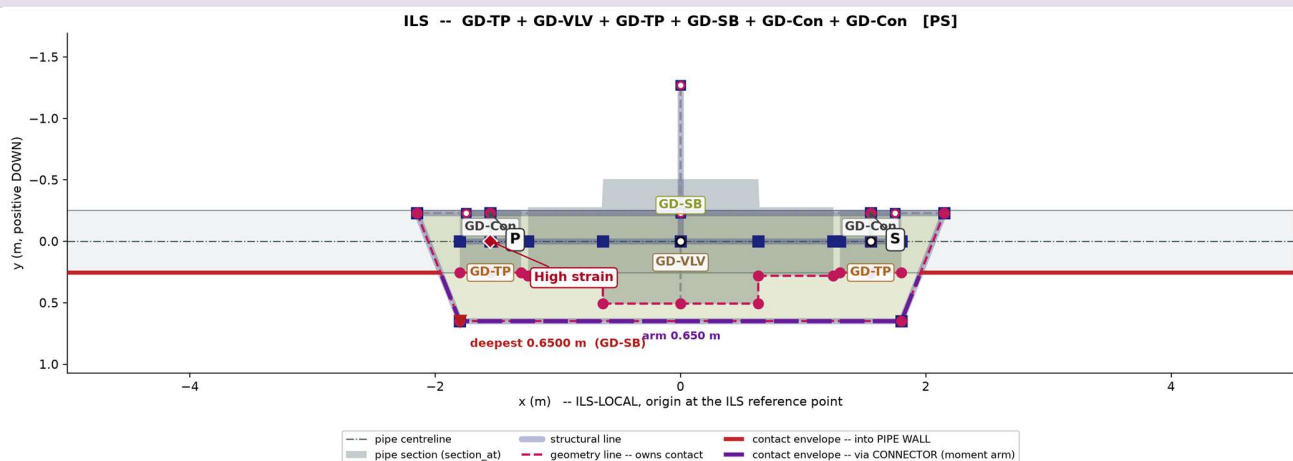


Figure 10.2. Accepted implementation Test 2 plot: inline valve protected by fitted GD-SB base.

Step 5 User response acceptance and correction note

The user accepted the GD-SB depth, GD-TP landing bodies, connector locations, and the final plot. During review, repeated P and S labels were identified as a plotter issue rather than an assembly build error. The plotter was corrected to show one connection label per physical GD-Con connector.

Step 6 LLM response record creation

The LLM recorded the accepted case

11. Professional Engineering Practice Relevance

The framework is designed around engineering practice rather than only AI novelty. Its value to a company would come from:

- making past design experience searchable and comparable;
- giving engineers a structured way to query historical evidence;
- making assumptions and missing inputs visible;
- preventing unsupported design sketches from being treated as engineering evidence;
- preserving review feedback in a reusable form;
- identifying whether a failure is due to workflow, ontology, retrieval, evidence, plotting, or missing study data;
- providing a path from historical knowledge to parametric studies and ML without losing human review.

For a practicing structural engineer, this is important because design intuition is often learned through exposure to many cases and corrections. KEL provides a mechanism for turning those corrections into structured knowledge. The LLM does not "develop intuition" in the human sense. Instead, the system develops a governed record of experience, failure modes, evidence limits, and updated rules that can support future reasoning.

This distinction is useful in professional communication. The work demonstrates an integrated capability across subsea structural engineering, ontology design, knowledge graphs, Python tooling, LLM workflow design, validation gates, plotting/reporting, and human-in-the-loop governance.

12. Conclusions

This case study frames engineering AI as governed knowledge work. The LLM interprets language, proposes problem records, plans retrieval, and explains results. The toolchain defines components, assemblies, evidence, gates, plots, feedback records, and lifecycle controls. Engineers confirm intent, judge evidence applicability, approve knowledge changes, and retain responsibility for project verification.

For S-lay inline structures, the design object is an interacting assembly. Component identity, topology, connection systems, support/protection assumptions, behavior evidence, and response limits must be explicit before an AI-assisted recommendation can be trusted. The future path continues the original ILS research programme: governed knowledge graphs plus hybrid retrieval, symbolic grounding, deterministic plotting and tools, KEL root-cause learning, parametric FEA studies, physics-informed ML, uncertainty-aware surrogate models, and engineering review.

Data, Software, and Reproducibility Statement

Supporting workflow instructions, toolchain records, and implementation notes are available from the corresponding author upon reasonable request.

AI-Assistance Statement

AI assistance was used to organize, draft, and revise this manuscript. The named authors remain responsible for verifying every citation, engineering statement, numerical value, figure, interpretation, and final submission. AI-generated text and diagrams are not treated as engineering evidence.

References

- [1] J. S. Gero, "Design Prototypes: A Knowledge Representation Schema for Design," *AI Magazine*, vol. 11, no. 4, pp. 26-36, 1990. doi:10.1609/aimag.v11i4.854.

- [2] A. K. Goel, S. Rugaber, and S. Vattam, "Structure, Behavior, and Function of Complex Systems: The Structure-Behavior-Function Modeling Language," *AI EDAM*, vol. 23, no. 1, pp. 23-35, 2009. doi:10.1017/S0890060409000080.
- [3] X. Fiorentini, I. Gambino, V. C. Liang, S. Foufou, S. Rachuri, M. Mani, and C. E. Bock, "An Ontology for Assembly Representation," NISTIR 7436, National Institute of Standards and Technology, 2007. doi:10.6028/NIST.IR.7436.
- [4] M. Dinar, A. Danieleescu, C. MacLellan, J. J. Shah, and P. Langley, "Problem Map: An Ontological Framework for a Computational Study of Problem Formulation in Engineering Design," *Journal of Computing and Information Science in Engineering*, vol. 15, no. 3, 031007, 2015. doi:10.1115/1.4030076.
- [5] Y. Li, H. Wang, B. Xue, M. Zhang, and Y. Jin, "Solver-Independent Automated Problem Formulation via LLMs for High-Cost Simulation-Driven Design," *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2138-2153, 2026. doi:10.18653/v1/2026.findings-acl.102.
- [6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W.-T. Yih, T. Rocktaschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems*, 2020. doi:10.48550/arXiv.2005.11401.
- [7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," *International Conference on Learning Representations*, 2023.
- [8] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," arXiv:2302.04761, 2023. doi:10.48550/arXiv.2302.04761.
- [9] Y. Xu, J. Xie, X. Liu, H. Cui, and M. Liu, "A Human-in-the-Loop Conceptual Design Framework Jointly Driven by Large Language Models and Knowledge Graphs," *Advanced Engineering Informatics*, vol. 74, part A, 104646, 2026. doi:10.1016/j.aei.2026.104646.
- [10] S. M. Sivaraman and J. V. Reddy, "Toward AI-Assisted Conceptual Design of Subsea Inline Structures: Classification and Strain Behaviour of Inline Components During S-Lay Installation," *IJRASET*, 2026.
- [11] S. M. Sivaraman and J. V. Reddy, "Advancing AI Assisted Conceptual Design of Subsea Inline Structures: Mechanical Behaviour of Pipeline-Mounted Structural Assemblies during S-Lay Installation," *IJRASET*, 2026.