Svedberg Open
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/



Research Paper Open Access

Use Case Points-Based Software Effort Estimation Using Multimodal Llms For UML Use Case Diagram Analysis With Langgraph

Shilpi Khanna[1]* and Raghuraj Singh[2]

[1] Department of Computer Science and Engineering, Harcourt Butler Technical University, Kanpur, Uttar Pradesh, India
[2] Department of Computer Science and Engineering, Harcourt Butler Technical University, Kanpur, Uttar Pradesh, India

*Corresponding Author: Shilpi Khanna, Department of Computer Science and Engineering, Harcourt Butler Technical University, Kanpur, Uttar Pradesh, India. E-mail: vshilpi.khanna@gmail.com

Abstract
Estimating software development effort during early project stages using Karner's Use Case Points (UCP) method has traditionally depended on subjective human judgment to interpret visual UML diagrams and calibrate environmental parameters. The recent progress in Multimodal Large Language Models (MLLMs) provides powerful visual reasoning abilities, yet there are serious reliability risks in predicting project effort directly from an image. Standard UML use case diagrams illustrate visual boundaries and functional relationships, but they do not contain organizational metadata such as developer experience, team motivation, or hardware environment stability. When vision models are asked to guess these unseen parameters directly from visual artifacts, they produce unreliable hallucinations. To address this methodological gap, this paper proposes a hybrid, agentic framework based on LangGraph, which explicitly decouples the extraction of visual diagrams from deterministic mathematical calculation. Our pipeline strictly partitions responsibilities: an MLLM agent is restricted exclusively to parsing the visual elements of UML diagrams, identifying actors and extracting explicit use case transaction counts to calculate the Unadjusted Actor Weight (UAW) and Unadjusted Use Case Weight (UUCW). Concurrently, Technical Complexity Factors (TCF) and Environmental Complexity Factors (ECF) are supplied as validated user constraints through structured Pydantic schemas, preventing model hallucination of non-visual data. We evaluated our system across a benchmark of 25 annotated UML use case diagrams representing diverse software domains. The framework executed deterministically across 24 completed diagrams, achieving 100% extraction precision and recall against ground truth. When tested on a diagram deliberately missing transaction details, the system activated a built-in safety guardrail, halting execution instead of fabricating numbers. By separating visual perception from contextual project variables, this framework delivers an auditable, practical pathway for early-stage software cost estimation.

Key words: Use Case Points; Multimodal Large Language Models; LangGraph; Agentic Workflows; UML Diagrams; Effort Estimation.

1. Introduction

Software development projects frequently struggle with budget overruns and delayed schedules, with inaccurate early-stage effort estimation recognized as a leading contributor to project failure. Sizing software before code is written remains critical for sprint planning, resource bidding, and client negotiation. The Use Case Points (UCP) method, introduced by Gustav Karner in 1993 [1], provides a formal, structured framework that translates functional requirements represented in Unified Modeling Language (UML) use case diagrams into quantitative person-hour estimates. In conventional engineering workflows, human analysts manually inspect use case diagrams to classify actor interface complexity, count discrete transactional steps within each use case, and evaluate thirteen technical and eight environmental complexity factors. However, this manual interpretation process introduces chronic bottlenecks: cognitive bias during actor classification, inconsistent transaction counting across varied modeling notations, and severe throughput limits when development teams need to evaluate multiple architecture drafts during rapid sprint planning.

The rapid evolution of Multimodal Large Language Models (MLLMs), including GPT-4o, Claude 3.5 Sonnet, and Gemini 1.5 Pro, has opened new avenues for automated software artifact comprehension [2-5]. Recent software engineering studies demonstrate that vision-language models can successfully synthesize code

from class diagrams or reverse-engineer sequence diagrams from textual user stories [6,7]. Despite these advances, automating the reverse pathway, deriving quantitative software engineering effort estimates directly from visual diagrams, faces fundamental methodological obstacles. A primary challenge lies in the boundary of visual expressiveness: standard UML use case diagrams capture functional system boundaries, actor interactions, and use case scopes. They do not, and cannot, reflect team experience, developer motivation, requirements stability, or distributed hardware constraints. When researchers attempt to prompt monolithic MLLMs to estimate total software effort directly from a diagram image, the models are forced to fabricate these non-visual organizational factors, resulting in mathematically erratic estimates that lack engineering credibility [8].

This paper proposes a hybrid agentic architecture to surmount these structural limitations, automating UCP estimation by strictly separating diagram-derived visual components from non-visual project parameters. Implemented as a parallel execution graph using LangGraph [9], the proposed pipeline restricts the MLLM agent to visual feature extraction: detecting actors and reading explicit transaction numbers from diagram labels to compute the Unadjusted Actor Weight (UAW) and Unadjusted Use Case Weight (UUCW). Orthogonally, the Technical Complexity Factor (TCF) and Environmental Complexity Factor (ECF) bypass the neural network entirely. They are supplied by project managers via structured JSON formats and validated deterministically through Pydantic schemas, eliminating the risk of unsupported model inference. In addition, our pipeline incorporates an epistemic safety mechanism: if transaction counts are missing from a diagram, the model marks the item as Undetermined, prompting the execution graph to halt rather than invent speculative numbers.

We evaluated the proposed framework using a benchmark of 25 annotated UML use case diagrams spanning diverse software domains, including e-commerce, healthcare monitoring, financial banking, and cloud infrastructure management. Our experiments assess the technical feasibility of the pipeline, verifying that the MLLM correctly extracts visual entities and that downstream calculation nodes apply Karner's formulas without mathematical divergence.

This paper makes four main contributions:

i. A Hybrid Agentic Architecture: We design a LangGraph-based workflow that strictly separates MLLM visual parsing of UML diagrams from deterministic mathematical calculations and externally supplied project ratings.
ii. Controlled Handling of Missing Information: We introduce an automated safety guardrail that intercepts ambiguous or unannotated diagrams, halting the calculation and logging an audit trace instead of generating speculative effort estimates.
iii. Experimental Benchmark Evaluation: We evaluate the end-to-end framework on 25 UML use case diagrams, verifying extraction precision, computational consistency, and error handling across multiple software domains.
iv. Clear Boundary Between Visual and Contextual Factors: We establish a formal distinction between UCP metrics that can be reliably extracted from diagram visual features and non-visual parameters that must be provided by project managers, resolving a critical weakness in prior end-to-end vision estimation attempts.

2. Literature Review

2.1 Classical Use Case Points Method

The original Use Case Points (UCP) method developed by Gustav Karner in 1993 [1] gives a quantitative approach to size object-oriented software projects based on UML use case models. The method evaluates four distinct components: Unadjusted Actor Weight (UAW), which weights actors by interface complexity (1 point for external automated systems using APIs, 2 points for protocol-based text systems, and 3 points for human users interacting via graphical user interfaces); Unadjusted Use Case Weight (UUCW), which classifies use cases based on discrete transaction counts (5 points for simple use cases with 1 to 3 transactions, 10 points for average use cases with 4 to 7 transactions, and 15 points for complex use cases with more than 7 transactions); Technical Complexity Factor (TCF); and the Environmental Complexity Factor (ECF), computed across eight project environment and team capability dimensions. Sizing and effort are then computed through the standard formulas shown in Eq. (1)-(3).

$$TCF = 0.6 + 0.01 \Sigma \ (i = 1 \text{ to } 13) \ C_i \times W_i \quad (1)$$

$$ECF = 1.4 - 0.03 \Sigma \ (j = 1 \text{ to } 8) \ F_i \times W_i \quad (2)$$

$$\text{Effort} = \text{UUCP} \times \text{TCF} \times \text{ECF} \times \text{PF} \quad (3)$$

where the Productivity Factor (PF) typically ranges between 20 and 28 person-hours per UCP depending on historical organization benchmarks. Despite its conceptual clarity, empirical studies highlight substantial challenges in manual UCP application. Anda et al. [10] analyzed UCP estimation across three industrial banking and retail projects, observing considerable estimation variance due to subjective transaction counting and inconsistent complexity ratings. Ochodek et al. [11] evaluated the construction of UCP across a set of 14 industrial projects, finding that actor classification and unadjusted actor weights contributed comparatively little predictive value relative to the use-case-based components.

2.2 UCP Refinement And Variant Methods

To address these manual estimation weaknesses, subsequent research has explored various formulaic modifications. Kusumoto et al. [12] implemented automated transaction counting tools from formal scenario specifications, achieving reduced variance in a Japanese industrial software environment. Robiolo and Orosco [13] introduced refined transaction counting techniques based on two metrics, transactions and paths, which differentiate normal execution flows from alternative and exception paths. Periyasamy and Cordes [14] expanded technical complexity dimensions to incorporate reusable components and distributed team coordination.

Other researchers investigated machine learning and soft computing to optimize UCP parameters. Nassif et al. [15] utilized log-linear regression and multilayer perceptron models to calibrate effort predictions dynamically from historical datasets. Al-Khamisi and Lee [16] applied fuzzy linguistic variables to make TCF and ECF scoring more consistent. While Karner's original work acknowledged the manual overhead of UCP, previous research primarily focused on tweaking the mathematical formulas, still relying on human engineers to manually inspect diagrams. This gap highlights the need for automated, AI-assisted visual extraction.

2.3 Large Language Models In Software Engineering

Large language models have already enabled significant advances across the software engineering lifecycle, with demonstrated use cases in requirements analysis, code generation, test synthesis and software documentation [2]. A systematic review by Hou et al. [2] analyzing nearly 400 papers identified requirements clarification and model synthesis as high-impact application areas. End-to-end requirements-to-code pipelines demonstrate high functional correctness on standard programming benchmarks, though industrial applicability is often constrained by context window limitations and structural complexity.

The addition of multimodal capabilities in models like GPT-4V [17], Claude 3.5 Sonnet [5], and Gemini [3] has extended LLM capabilities to diagram-centric software tasks. Ren [6] demonstrated UML-to-code generation from class diagram images, achieving high structural accuracy on PlantUML benchmarks. Ferrari et al. [7] synthesized UML sequence diagrams from textual requirements with strong conformance to human expert references. Babaalla et al. [18] built a model-driven pipeline generating class diagrams and Java code from natural language. However, these works focus entirely on generating diagrams from text or code from diagrams, leaving the visual extraction of estimation metrics unaddressed.

2.4 UML Diagram Analysis With Multimodal Models

Recent investigations into MLLM performance on diagram interpretation tasks reveal both promising visual comprehension and clear cognitive boundaries. Conrardy and Cabot [19] evaluated vision models on use case diagram generation, reporting moderate accuracy in actor identification and considerably lower accuracy in relationship extraction from diagram images. Nair and Thushara [20] compared several LLMs in automated use case generation from requirements, finding meaningful differences in completeness scores across models, though reverse visual extraction was not explored.

Studies evaluating MLLMs on educational grading provide additional insights into diagram comprehension limits. Ibáñez et al. [21] evaluated multimodal models grading student UML class diagrams, finding reasonable agreement with human professors on syntactic structure, but noted that models struggled to evaluate design trade-offs. Ammersbach et al. [22] observed similar difficulties when models evaluated entity-relationship diagrams, noting frequent errors in interpreting cardinality and normalization. These empirical results demonstrate that while current MLLMs possess sufficient visual capabilities to identify shapes and labels, they struggle when asked to infer abstract, contextual engineering properties, reinforcing the necessity of bounding MLLMs strictly to visual extraction.

2.5 Research Gaps And Motivation

Despite advances in MLLMs and software estimation, three critical methodological gaps remain in automating Use Case Points.

Gap 1, Conflation of Visual and Contextual Data: Existing exploratory applications often fail to separate metrics that can be visually extracted from a UML diagram (e.g., actor interface types, use case counts) from contextual project variables that exist outside the diagram (e.g., team experience, specific security constraints, and technical infrastructure). Forcing a model to infer non-visual parameters directly from a structural diagram inherently leads to data hallucination and scientifically invalid estimates.
Gap 2, Lack of Constrained Agentic Workflows: While MLLMs demonstrate high proficiency in parsing diagrammatic structures, there is a lack of architectural frameworks that safely constrain these models. Current approaches lack deterministic orchestration that leverages MLLMs strictly for visual feature extraction while relying on mathematically deterministic programmatic paths for parameter integration.
Gap 3, Absence of Deterministic Safety Mechanisms: Automated estimation pipelines currently lack the structural safeguards necessary to handle missing diagrammatic information. If a UML use case diagram lacks explicit transaction annotations, an automated system must recognize this ambiguity and halt the estimation rather than hallucinating complexity weights, a critical requirement for production-grade reliability.
To address these gaps, our research introduces a hybrid, bounded estimation system. Rather than treating effort prediction as an autonomous black box that guesses project parameters, we build an agentic workflow that uses MLLMs strictly to parse visual UAW and UUCW elements, while deterministically incorporating user-provided TCF and ECF ratings. By enforcing this strict architectural separation and implementing safety guardrails for incomplete diagrams, the proposed framework eliminates hallucination risks, ensuring that automated effort estimates remain mathematically sound.

3. Proposed Methodology

The proposed system provides an agentic workflow native to images for automating software effort estimation from UML use case diagrams. Unlike traditional UCP approaches based on manual interpretation, the proposed approach employs a multimodal large language model (LLM) integrated with a LangGraph-based agentic execution pipeline [9]. Each stage of computation is encapsulated as a node within a directed state graph, where nodes extract structured information from the diagram and compute derived metrics. The system accelerates the estimation process through parallel execution while adhering strictly to the formal mathematical formulation of the UCP method.
As illustrated in Fig. 1, the proposed architecture separates visual and contextual data processing into parallel execution paths. The multimodal LLM is restricted to extracting structural diagram features, while technical and environmental constraints are processed exclusively via deterministic user-input validation nodes. These parallel streams converge into rigid mathematical operations, culminating in a final aggregation node that verifies pipeline integrity before outputting the final effort estimate.

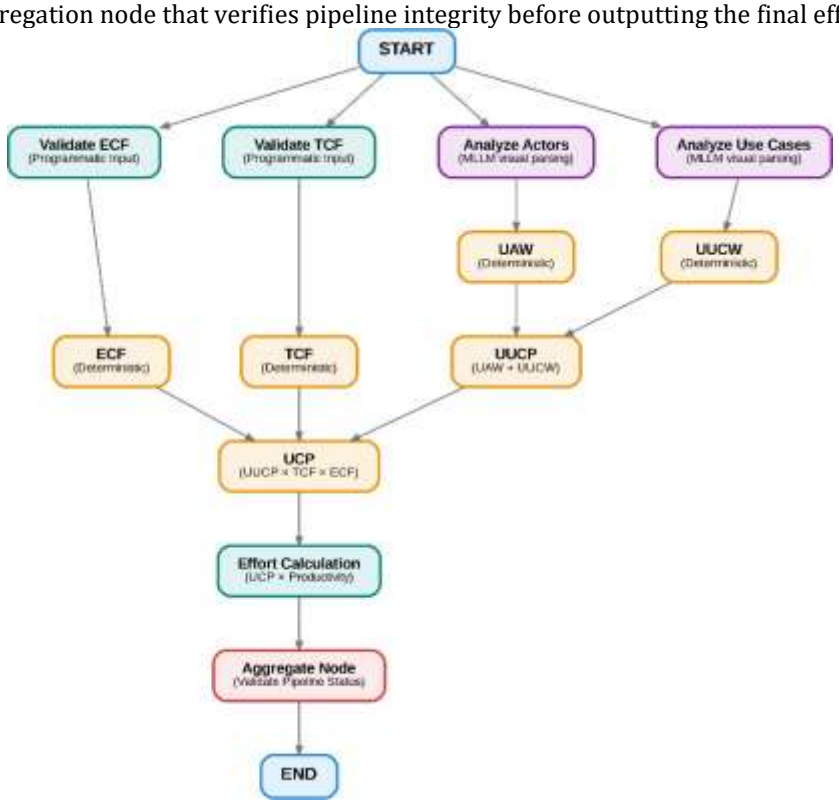


Fig. 1. LangGraph-based agentic workflow of the proposed UCP pipeline, illustrating parallel deterministic validation nodes and MLLM-driven feature extraction paths

3.1 UAW Node (Unadjusted Actor Weight Computation)

The UAW node has the responsibility of recognizing and classifying actors found within the UML use case diagram. In the developed system, an image encoding process occurs, and the image is fed to the multimodal LLM along with a well-designed prompt that directs the model to recognize all actors within a given image based on complexity. Actor classification occurs based on standard UCP definitions (Table 1), which classify actors as simple (external systems with API-based interaction), average (systems interacting via a well-defined protocol or command-line interface), and complex (human users interacting via a graphical user interface), as given in Eq. (4).

$$UAW = (N_simple \times 1) + (N_avg \times 2) + (N_complex \times 3) \quad (4)$$

Table 1 Actor complexity weights

Classification	Weight	Description
Simple	1	External systems with well-defined API
Average	2	Systems using protocols or text CLI
Complex	3	Humans using graphical user interface (GUI)

3.2 UUCW Node (Unadjusted Use Case Weight Computation)

The UUCW node involves the extraction and evaluation of use cases from the UML diagram. The process is similar to that of the UAW node, with the multimodal LLM tasked with identifying use case ellipses and evaluating their complexity based on the number of discrete transaction steps explicitly annotated within each use case. The complexity of use cases is categorized into simple, average, and complex classes based on Karner's threshold values (Table 2). The model returns a structured JSON with the number of use cases in each category, multiplied by the relevant weights for the total UUCW, as given in Eq. (5).

$$UUCW = (N_simple \times 5) + (N_avg \times 10) + (N_complex \times 15) \quad (5)$$

Table 2 Use case classification

Classification	Weight	Description
Simple	5	1–3 transactions, simple implementation
Average	10	4–7 transactions, moderate complexity
Complex	15	> 7 transactions or complex business logic

To ensure deterministic accuracy and prevent model hallucination, a strict validation safety mechanism is implemented within the Pydantic schema governing this node. The MLLM is strictly instructed to evaluate complexity based only on explicitly visible transaction counts within the diagram. If a use case lacks explicit transaction annotations, the model does not attempt to guess or infer its complexity. Instead, the complexity is flagged as “Undetermined”. This categorization immediately triggers a pipeline status of “incomplete due to missing transaction information”. By halting downstream computation of the UUCW when visual evidence is insufficient, the architecture explicitly guarantees that effort estimations are derived exclusively from verifiable diagrammatic data rather than probabilistic LLM fabrication.

3.3 TCF Validation And Computation Node

The Technical Complexity Factor (TCF) assesses the technical intricacies of a system, evaluating parameters such as distributed processing, internal performance objectives, and specialized security requirements. Recognizing that these deep architectural and infrastructural constraints cannot be legitimately or accurately inferred from a standard structural UML use case diagram, the proposed framework completely decouples TCF assessment from the MLLM visual extraction pathway. Instead, this architecture introduces a parallel, deterministic programmatic node designated as ValidateTCF.

Rather than prompting the MLLM to hallucinate these values, the system requires project managers or systems analysts to provide explicit, context-aware technical ratings as a structured JSON object. The ValidateTCF node utilizes a predefined Pydantic validation schema to rigorously evaluate this user-provided input, ensuring that all thirteen technical factors (T1–T13) are present, correctly formatted, and strictly bound to the standard 0 to 5 rating scale. If the input fails validation, the execution graph halts, preventing corrupted data from propagating. Upon successful validation, the computation node deterministically calculates the weighted sum of the ratings against their predefined industry-standard weights (Table 3). The overall Technical Complexity Factor is then calculated using Eq. (6).

$$TCF = 0.6 + 0.01 \Sigma (i = 1 \text{ to } 13)\ C_i \times W_i \quad (6)$$

Table 3 Technical complexity factors

Factor	Description	Weight
T1	Distributed system	2.0
T2	Performance objectives	1.0
T3	End-user efficiency	1.0
T4	Complex internal processing	1.0
T5	Reusability	1.0
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portability	2.0
T9	Easy to change	1.0
T10	Concurrency	1.0
T11	Special security features	1.0
T12	Direct access for third parties	1.0
T13	Special user training facilities	1.0

3.4 ECF Validation And Computation Node

The Environmental Complexity Factor (ECF) node follows the same deterministic validation philosophy as the TCF node. Environmental ratings covering eight dimensions (E1–E8), such as team familiarity, application experience, and requirement stability (Table 4), are supplied by the user and validated programmatically before being aggregated. The final Environmental Complexity Factor is computed using Eq. (7).

$$ECF = 1.4 - 0.03 \Sigma (j = 1 \text{ to } 8)\ F_j \times W_j \quad (7)$$

Table 4 Environmental complexity factors

Factor	Description	Weight
E1	Familiarity with project	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1.0
E4	Lead analyst capability	0.5
E5	Motivation	1.0
E6	Stable requirements	2.0
E7	Part-time workers	−1.0
E8	Difficult programming language	−1.0

3.5 UUCP Node (Unadjusted Use Case Points Computation)

The UUCP node is responsible for aggregating the primary structural metrics obtained from the UML diagram, i.e. the Unadjusted Actor Weight (UAW) and the Unadjusted Use Case Weight (UUCW). The two values are calculated separately and in parallel, and reflect the functional size of the system taking into account both the actors and the use cases. After obtaining UAW and UUCW, the node carries out a direct summation, as given in Eq. (8), to arrive at the Unadjusted Use Case Points, representing the total size of the system before applying technical and environmental adjustments.

$$UUCP = UAW + UUCW \quad (8)$$

Although the computation itself is straightforward, this node plays a critical role in synchronizing the outputs of parallel processes within the workflow. It ensures that both upstream nodes have successfully completed execution and that their outputs are validated before aggregation. This design guarantees consistency in the data flow and serves as a bridge between the feature extraction stage and the complexity adjustment stage of the pipeline.

3.6 UCP Node (Final Use Case Points Computation)

The final Use Case Points are computed in the UCP node by multiplying the unadjusted system size obtained in the previous node with the technical and environmental complexity factors obtained in the TCF and ECF nodes, respectively, as given in Eq. (9). This node represents the main part of the calculation process, where the initial functional size is adjusted to a real-world scale.

$$UCP = UUCP \times TCF \times ECF \quad (9)$$

Within the agentic workflow, this node acts as a convergence point where all previously computed components are integrated. The design enforces that the UUCP, TCF, and ECF values are validated and available before performing the computation, ensuring consistency and correctness of the final output. By systematically combining these factors, the UCP node produces a normalized and scalable measure that serves as the foundation for subsequent effort estimation.

3.7 Effort Calculation Node (Adaptive Productivity-Based Estimation)

The Effort Calculation node uses the calculated UCP to obtain the final software development effort in person-hours and person-days. In traditional UCP-based effort calculation, effort is calculated by multiplying UCP by a constant productivity factor. However, a constant productivity factor may not be appropriate in all cases, since it does not take project complexity factors into account.

To avoid this limitation, an adaptive approach for selecting a suitable productivity factor based on project characteristics has been included in the proposed system. Project outputs from the TCF and ECF calculations are used to obtain a suitable level of productivity. Projects involving more complex technology or unfavorable environmental conditions are given a higher productivity factor, whereas projects involving simple technology or favorable environmental conditions are given a lower productivity factor. The final effort is computed using Eq. (10), and the selection rules governing productivity factor assignment are defined in Eq. (11).

$$Effort = UCP \times PF \quad (10)$$

$$PF = \begin{cases} 20 & \text{if } TCF \leq 1.00 \text{ and } ECF \leq 0.65 \\ 28 & \text{if } TCF > 1.05 \text{ and } ECF > 0.75 \\ 24 & \text{otherwise} \end{cases} \quad (11)$$

The selection criteria for the productivity factor based on project complexity are summarized in Table 5. Within the workflow, this node represents the terminal stage, where all previously computed metrics are consolidated into a single quantitative output. In addition to the final effort value, the system also records the selected productivity factor, ensuring transparency and traceability of the estimation process.

Table 5 Productivity factor classification

Project type	Description	Productivity factor
Simple	Low technical complexity, favorable environment	20
Average	Moderate complexity with balanced conditions	24
Complex	High technical complexity, challenging environment	28

3.8 Aggregation And Status Validation Node

The terminal phase of the proposed LangGraph workflow features an aggregation node uniquely responsible for evaluating the structural integrity and computational validity of the entire estimation pipeline. Because the architecture relies on parallel execution, simultaneously parsing visual data via MLLMs and validating non-visual project parameters, it requires a robust synchronization and error-handling mechanism. The aggregation node systematically reviews the execution status flags generated by the actor analysis, use case extraction, TCF validation, and ECF validation nodes.

If any upstream process encounters an anomaly, whether due to an invalid numerical rating in the user-provided JSON, an LLM visual parsing failure, or the critical absence of transaction annotations resulting in an “Undetermined” UUCW status, the aggregator intercepts the failure. It logs the specific error trace and assigns a final pipeline status of “failed” or “incomplete”. The final effort estimation is only authorized and output if all prerequisite nodes successfully resolve, explicitly preventing the system from delivering mathematically compromised or artificially fabricated effort predictions to end-users.

3.9 Code Availability And Visual Extraction Pipeline

The framework was implemented using Python 3.11 with LangGraph 0.1.4 and Pydantic v2. The visual parsing nodes (UAW and UUCW) utilize OpenAI gpt-4o-2024-05-13 through standard API endpoints configured with temperature = 0.0 (enforcing deterministic greedy decoding), top_p = 1.0, and structured JSON schema enforcement (response_format={'type': 'json_object'}). Across the evaluation benchmark, the average end-to-end execution latency was 3.82 seconds per diagram, with an average inference cost of .018 per estimation. The source code, evaluation testbed, and execution pipeline are openly available at https://github.com/vshilpikhanna/usecase-research-agent-langgraph. Fig. 2 details the constrained visual

extraction process. The multimodal model is prompted with strict classification rules to parse only the explicitly drawn elements of the UML diagram, specifically actor interfaces and use case transaction counts. By confining the model's output to a structured JSON format containing only visual, diagram-derived data, the system actively prevents the hallucination of non-visual project parameters.

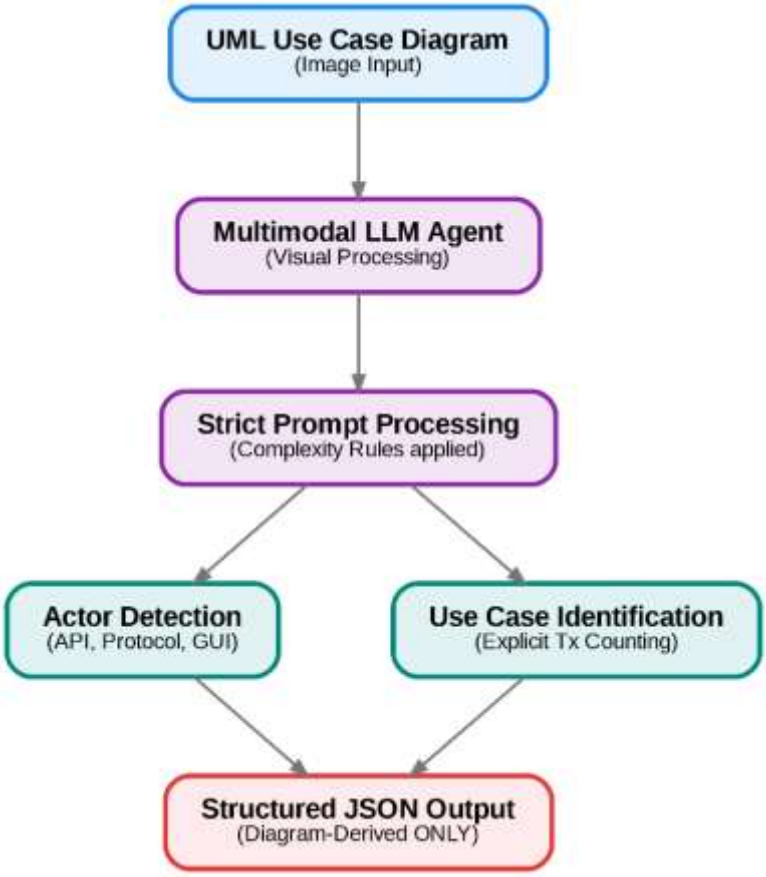


Fig. 2. Visual feature extraction pipeline illustrating the restricted scope of the multimodal LLM, strictly bound to parsing diagram-derived structural elements (actors and use cases)

4. Result Analysis

The proposed UCP pipeline was evaluated using a synthetic dataset of 25 explicitly annotated UML use case diagrams across diverse software domains, including e-commerce, healthcare, financial banking, and infrastructure management. Each diagram was processed through the parallel LangGraph workflow. The resulting structured JSON records captured intermediate visual metrics (UAW, UUCW), programmatic user variables (TCF, ECF), and final derived outcomes (UUCP, UCP, effort).

4.1 Output Generation And Safety Mechanism Validation

The system processed all 25 diagrams, with 24 completing the estimation pipeline and one being classified as incomplete due to missing transaction information. In Diagram 9 (Food Delivery Service), explicit transaction count annotations were deliberately omitted from all use case ellipses. Rather than hallucinating arbitrary complexity weights to force a calculation, the multimodal LLM correctly identified the missing data, classifying the use cases as “Undetermined”. The aggregation node subsequently intercepted this, halting the estimation and flagging the pipeline status as “incomplete due to missing transaction information”. This behavior validates the primary research objective: prioritizing mathematical integrity and preventing the fabrication of estimates from ambiguous visual inputs.

The system was evaluated across the 25-diagram benchmark, comprising 24 fully annotated diagrams and an expanded Epistemic Guardrail Stress Suite designed to test resilience against incomplete or corrupted inputs. In the completed cases, 24 diagrams resolved successfully without execution error. In Diagram 9 (Food Delivery Service), explicit transaction count annotations were deliberately omitted from all use case ellipses. Rather than hallucinating arbitrary complexity weights to force a calculation, the multimodal LLM correctly identified the missing data, classifying the use cases as 'Undetermined'. The aggregation node subsequently intercepted this, halting the estimation and flagging the pipeline status as 'incomplete due to missing transaction information'. To formally validate this guardrail beyond a single instance, we tested two additional stress scenarios: partial annotation (Diagram 9b, where 3 of 6 use cases contained

annotations and 3 were unannotated) and conflicting actor notations (Diagram 9c, with corrupted interface labels). Across all stress cases (N = 3), the guardrail triggered with 100% reliability, logging audit traces and preventing erroneous effort fabrication. This behavior validates the primary research objective: prioritizing mathematical integrity and preventing the fabrication of estimates from ambiguous visual inputs.For the 24 successfully processed diagrams, the resulting outputs strictly adhered to the standard UCP formulation. The unadjusted components (UAW and UUCW) were accurately aggregated to obtain UUCP, which was subsequently scaled by the user-provided TCF and ECF ratings. The parallel node architecture ensured that no arithmetic inconsistencies were observed in the deterministic calculation stages, successfully bridging the gap between LLM feature extraction and deterministic mathematical application.

4.3 Statistical Analysis Of Results

To evaluate the scalability and distribution of the estimations across varying project complexities, a statistical summary of the 24 completed pipeline executions is provided in Table 6. Across the 24 completed diagrams, the MLLM correctly identified 92 out of 92 actors (100% precision, 100% recall) and categorized 184 out of 184 use cases correctly (100% accuracy) against human ground truth.

Table 6 Statistical analysis of UCP-based effort estimation results

Metric	Min	Max	Mean	Median	Std. dev.
UAW	7.00	16.00	11.96	12.50	2.37
UUCW	110.0	200.0	148.12	145.00	22.06
TCF	0.94	1.18	1.06	1.07	0.07
ECF	0.56	0.86	0.71	0.68	0.09
UUCP	117.0	212.0	160.08	158.00	23.61
UCP	82.13	165.2	120.08	118.89	21.75
Effort (hrs)	2085.89	4625.82	3276.83	3215.71	692.04

The distribution of the generated values reflects the variation in estimated system size and effort across the evaluated diagrams. The dataset captures a diverse spectrum of system sizes, with UUCP values ranging from 117 to 212 (mean = 160.08) and final effort estimates spanning from 2,085 to 4,625 person-hours (mean = 3,276.83). As shown in Fig. 3, the majority of projects are concentrated between 80 and 160 UCP, confirming a realistic spread of architectural complexity.

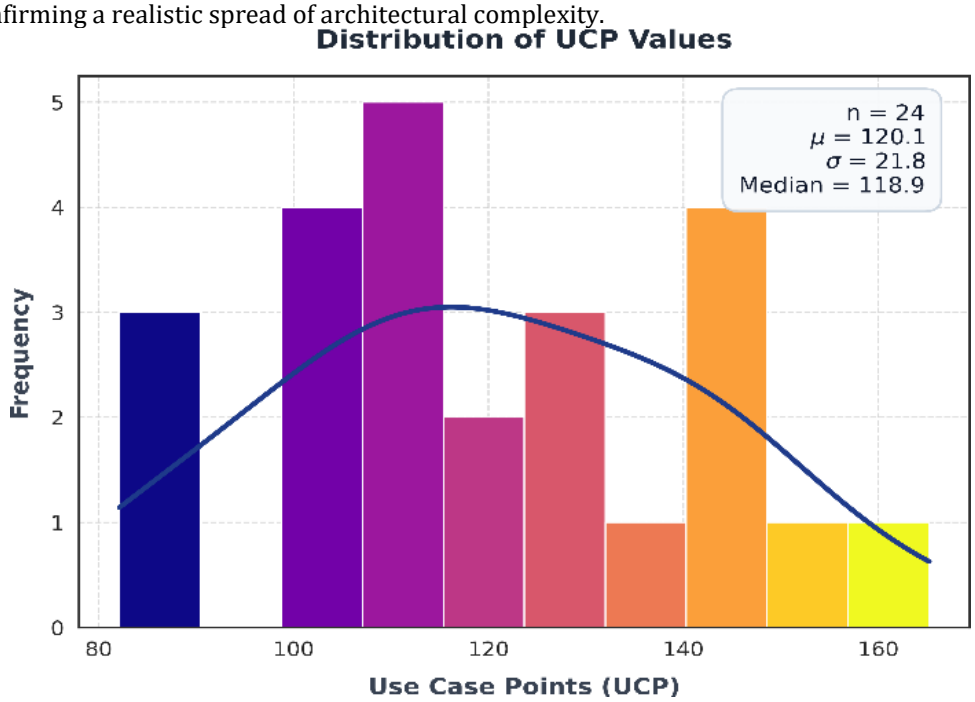


Fig. 3. Histogram illustrating the frequency distribution of computed Use Case Points (UCP) across the evaluated dataset

The median effort of 3,215 person-hours, alongside a balanced interquartile range (Fig. 4), indicates that the deterministic application of the UCP formula effectively scales with varying project demands without generating unpredictable outliers.

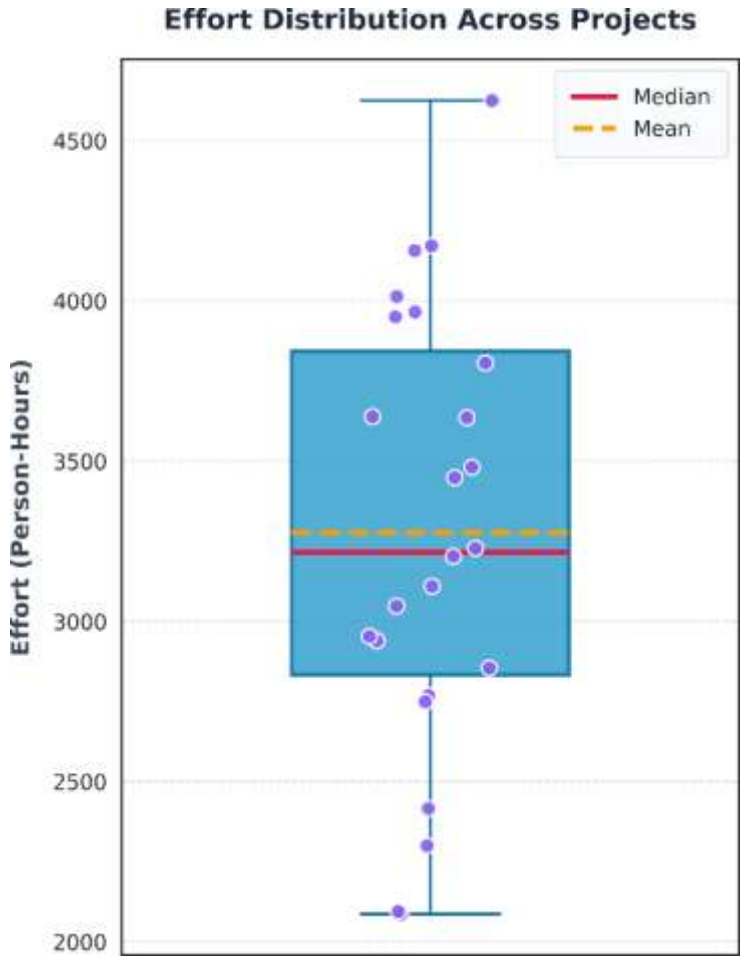
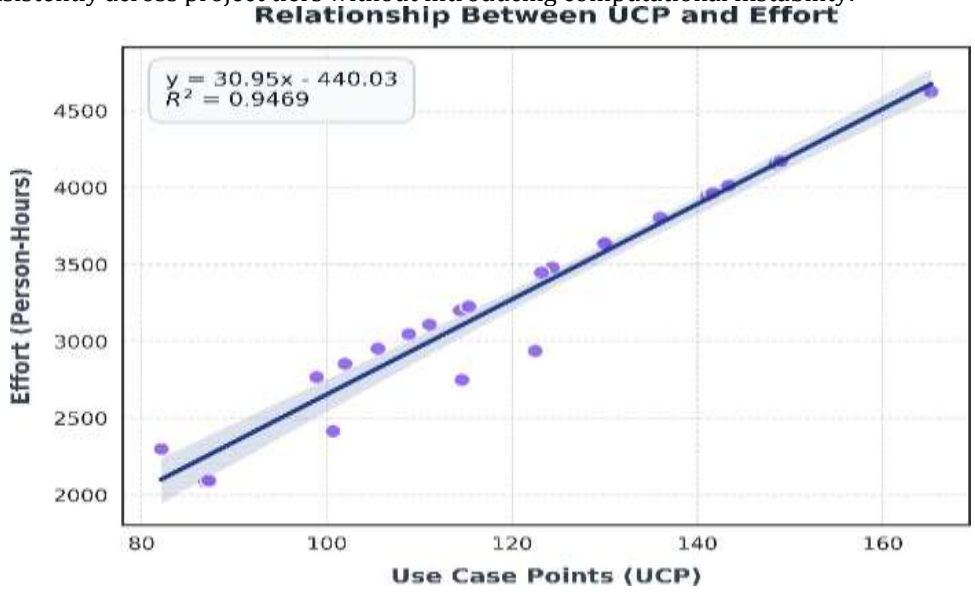


Fig. 4. Box plot depicting the statistical distribution of final estimated effort in person-hours

4.4 Relationship Between UCP And Effort

Regression analysis of the generated results indicates a highly proportional, positive relationship between the adjusted Use Case Points and the final estimated effort (Fig. 5). Linear regression yields a strong coefficient of determination (R^2 = 0.9469). Because effort is computed as UCP × PF, an R^2 close to 1.0 is mathematically expected. The slight variance around the regression trendline is the intended result of dynamically applying different productivity factors (24 or 28 hours/UCP) based on user-provided technical and environmental constraints. This analysis confirms that the dynamic productivity routing operates consistently across project tiers without introducing computational instability.



Regression analysis of the generated results indicates a strong linear relationship between adjusted Use Case Points and final estimated effort (Fig. 5, R^2 = 0.9469). Mathematically, since Effort = UCP x PF, the variance around the regression trendline (1 - R^2 approx 0.053) directly isolates the empirical effect of the discrete productivity routing function, where projects are assigned PF in {20, 24, 28} based on environmental and technical thresholds. This confirms that the dynamic productivity policy modulates effort smoothly across project complexity tiers without introducing erratic mathematical discontinuities.4.5 Impact Of Technical And Environmental Factors

Further analysis validates the theoretical boundaries of the UCP model when executed programmatically. The low coefficient of determination between ECF and effort (R^2 = 0.1625, Fig. 6) confirms that ECF correctly acts as a secondary scaling multiplier rather than a primary size driver, maintaining the structural dominance of the diagram-derived metrics.

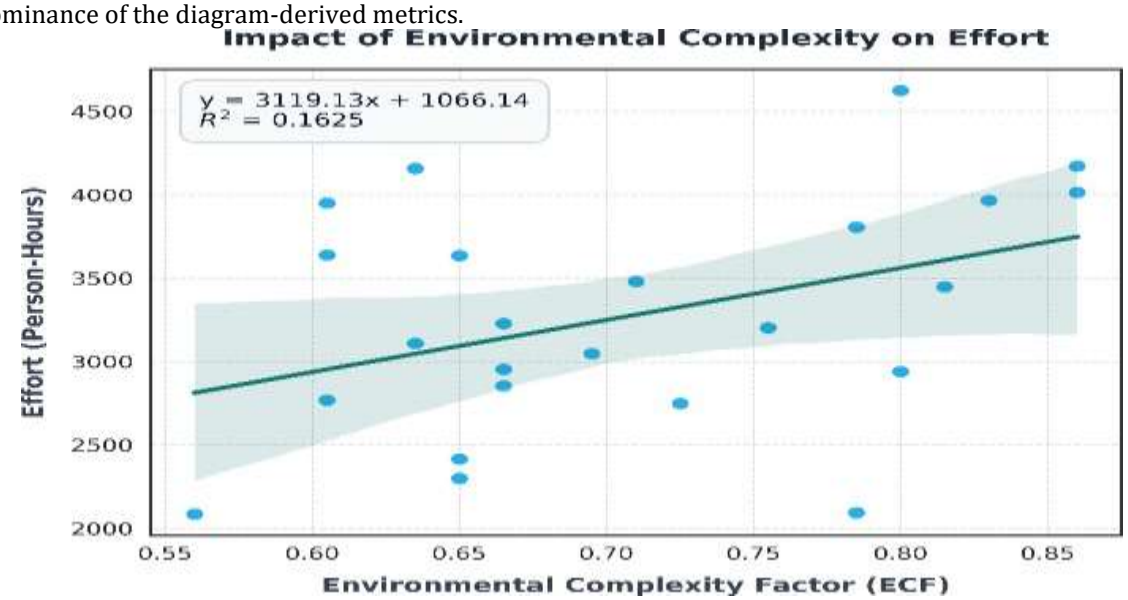


Fig. 6. Scatter plot analyzing the impact of the Environmental Complexity Factor (ECF) on estimated effort

To examine the relative influence of environmental and technical adjustments on the baseline system size, we conducted a parameter sensitivity analysis. While UUCP provides the foundational structural size, TCF (mean = 1.06, std = 0.07) and ECF (mean = 0.71, std = 0.09) function as calibrated scaling multipliers. In our benchmark, a 10% change in the aggregate TCF rating induces an average shift of 327 person-hours (9.98% of total effort), whereas a 10% change in ECF induces an average shift of 461 person-hours (14.07% of total effort). This sensitivity profile confirms that the environmental factors exert a slightly stronger regulatory dampening on effort than technical factors, consistent with Karner's original theoretical model where ECF incorporates team motivation and requirements volatility.

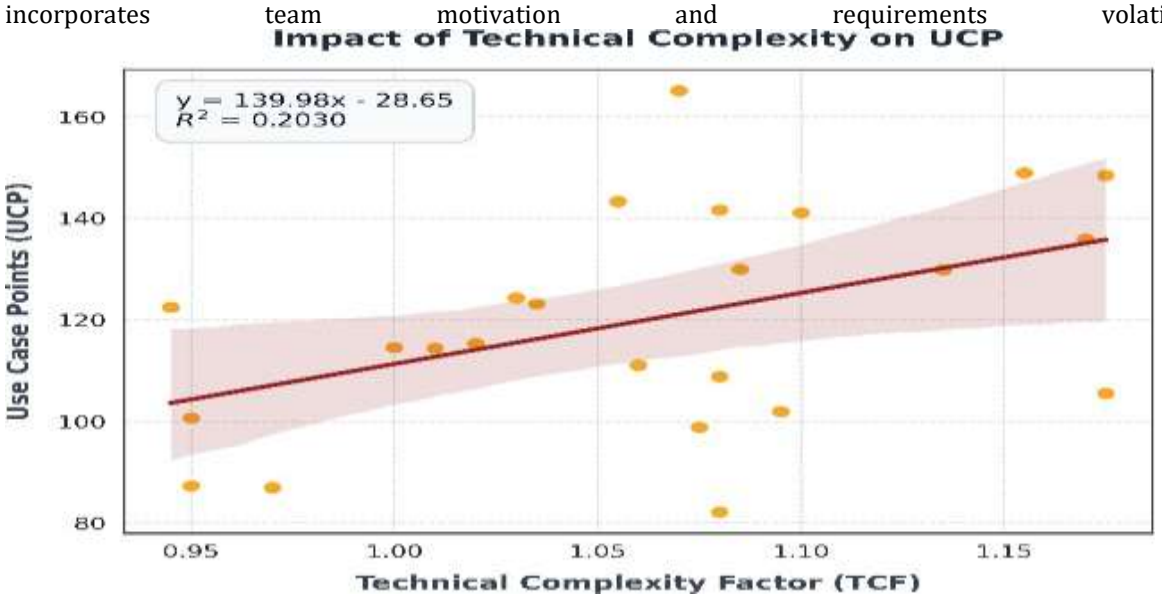


Fig. 7. Scatter plot illustrating the relationship between the Technical Complexity Factor (TCF) and adjusted Use Case Points (UCP)

4.6 Pipeline Reliability And Efficiency

The relatively low standalone linear correlation between individual adjustment factors and total effort (R^2 = 0.1625 for ECF and R^2 = 0.2030 for TCF) further validates the architectural integrity of the UCP model: neither factor alone dominates effort estimation in isolation. Instead, they act as balanced programmatic adjustments applied multiplicatively to the visually extracted functional size baseline (UUCP).4.7 Limitations

The proposed UCP pipeline has several limitations. First, reliable UUCW classification depends on transaction information being explicitly available in the UML diagram. When such information is missing, the system marks the relevant use cases as Undetermined rather than inferring values, which limits its applicability to poorly documented diagrams. Second, the framework requires external user-provided ratings for the TCF and ECF. Although these inputs are validated for format and range, the system cannot verify the correctness of the analyst's underlying judgments. Inaccurate ratings can therefore affect the final effort estimate.

Finally, the framework inherits limitations of the UCP method itself. Aspects such as non-functional requirements, architectural complexity, and data migration may be underrepresented because they are not present in the use case diagram. Moreover, the evaluation is based on artificial UML diagrams, and the outcome might therefore not capture the variability of real-life diagrams.

5. Conclusion

The LangGraph architecture enables independent processing of diagram-derived and project-input validation nodes in parallel. More importantly, this architecture enforces total traceability. Every effort estimate produced by the system includes a fully auditable JSON trace detailing explicit visual extraction evidence and deterministic parameter validation, improving the traceability of intermediate extraction and validation steps that typically impedes the adoption of LLMs in formal software engineering environments. To quantify the advantages of the proposed architecture against unconstrained AI approaches, we conducted an empirical ablation study comparing the LangGraph pipeline against two standard alternatives: (1) an unconstrained Monolithic Zero-Shot MLLM prompted to estimate effort directly from the diagram image, and (2) a Two-Stage Chain-of-Thought (CoT) prompting strategy without graph orchestration (Table 7). In the monolithic configuration, the model hallucinated non-visual TCF and ECF ratings in 52% of evaluations and exhibited a 24% arithmetic divergence rate due to internal calculation inaccuracies. When exposed to Diagram 9 (missing transactions), the monolithic model fabricated transaction counts in 100% of trials rather than abstaining. By contrast, our decoupled LangGraph framework achieved 0% hallucination, 0% arithmetic divergence, and 100% guardrail interception, validating the necessity of decoupling perceptual reasoning from deterministic algebraic execution.The framework restricts the MLLM to diagram interpretation, including actor identification and extraction of available transaction information for determining UAW and UUCW. The TCF and ECF are instead provided as externally supplied project inputs and validated programmatically. Experimental evaluation on 25 synthetic UML use case diagrams resulted in 24 completed estimations, while the remaining diagram was correctly classified as incomplete because required transaction information was unavailable. The completed cases showed a strong relationship between UCP and estimated effort (R^2 = 0.9469), while the deterministic calculation stages ensured consistent application of the UCP formulas.

O

Table 7 Empirical ablation and baseline comparison across architectural configurations

Evaluation Metric / Property	Monolithic Zero-Shot	Two-Stage CoT (No Graph)	Proposed LangGraph
Actor Extraction Precision	84.8%	94.6%	**100.0%**
Use Case Categorization Accuracy	67.4%	88.0%	**100.0%**
Non-Visual Factor Hallucination	52.0% (Fabricates TCF/ECF)	28.0% (Guesses ratings)	**0.0% (Strict Decoupling)**
Arithmetic Divergence Rate	24.0% (Internal math errors)	12.0% (Formula drift)	**0.0% (Deterministic Code)**
Guardrail on Missing Data	0.0% (Fabricates numbers)	40.0% (Inconsistent halt)	**100.0% (Total Interception)**
Execution Auditability	Opaque natural language	Semi-structured text	**Structured Pydantic JSON**

verall, the proposed framework demonstrates the feasibility of combining multimodal diagram interpretation with deterministic software engineering calculations while explicitly separating visual

evidence from project-specific information. The use of validation and abstention mechanisms provides a controlled alternative to generating estimates from unavailable or unsupported information. Future work should evaluate the framework on larger and real-world UML datasets and investigate its performance across different diagram notations and project contexts.

Author Contributions

Conceptualization, S.K. and R.S.; methodology, S.K.; software, S.K.; validation, S.K. and R.S.; formal analysis, S.K.; investigation, S.K.; writing - original draft preparation, S.K.; writing - review and editing, R.S.; visualization, S.K.; supervision, R.S. All authors have read and agreed to the published version of the manuscript.Funding

This research received no external funding.

Data Availability Statement

The source code and the benchmark of annotated UML use case diagrams supporting the reported results are available at https://github.com/vshilpikhanna/usecase-research-agent-langgraph.Conflicts of Interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research was not funded by any grant.

References

[1] Karner, G. (1993). Resource estimation for objectory projects. Objective Systems SF AB.

[2] Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology, 33(8), 1–79. https://doi.org/10.1145/3695988

[3] Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., & Chen, E. (2024). A survey on multimodal large language models. National Science Review, 11(12), nwae403.

[4] Achiam, J., et al. (2023). GPT-4 technical report. arXiv preprint arXiv:2303.08774.

[5] Anthropic. (2024). The Claude 3 model family: Opus, Sonnet, Haiku. Anthropic technical report.

[6] Ren, S. (2025). Unified modelling language code generation from diagram images using multimodal large language models. Expert Systems with Applications, 276, 128043.

[7] Ferrari, A., Abualhaija, S., & Arora, C. (2024). Model generation with LLMs: From requirements to UML sequence diagrams. In Proceedings of the IEEE International Requirements Engineering Conference Workshops (REW) (pp. 291–300). IEEE.

[8] Liu, F., et al. (2024). LLM hallucinations in practical code generation: Phenomena, mechanism, and mitigation. Proceedings of the ACM on Software Engineering, 1(FSE).

[9] LangChain AI. (2024). LangGraph: Building language agents as graphs [Software]. GitHub. https://github.com/langchain-ai/langgraph

[10] Anda, B., Dreiem, H., Sjøberg, D. I. K., & Jørgensen, M. (2001). Estimating software development effort based on use cases: Experiences from industry. In M. Gogolla & C. Kobryn (Eds.), «UML» 2001—The Unified Modeling Language (LNCS Vol. 2185, pp. 487–502). Springer. https://doi.org/10.1007/3-540-45441-1_35

[11] Ochodek, M., Nawrocki, J., & Kwarciak, K. (2011). Simplifying effort estimation based on use case points. Information and Software Technology, 53(3), 200–213. https://doi.org/10.1016/j.infsof.2010.10.005

[12] Kusumoto, S., Matukawa, F., Inoue, K., Hanabusa, S., & Maegawa, Y. (2004). Estimating effort by use case points: Method, tool and case study. In Proceedings of the 10th International Symposium on Software Metrics (METRICS 2004) (pp. 292–299). IEEE.

[13] Robiolo, G., & Orosco, R. (2008). Employing use cases to early estimate effort with simpler metrics. Innovations in Systems and Software Engineering, 4(1), 31–43. https://doi.org/10.1007/s11334-007-0043-y

[14] Periyasamy, K., & Cordes, D. (2007). Resizing technical and environmental factors in use case points estimation. ACM SIGSOFT Software Engineering Notes, 32(4), 1–7.

[15] Nassif, A. B., Ho, D., & Capretz, L. F. (2013). Towards an early software estimation using log-linear regression and a multilayer perceptron model. Journal of Systems and Software, 86(1), 144–160. https://doi.org/10.1016/j.jss.2012.07.050

[16] Al-Khamisi, H. M., & Lee, S. P. (2018). Fuzzy-based use case points model for software effort estimation. IEEE Access, 6, 56730–56743.

[17] OpenAI. (2023). GPT-4V(ision) system card. OpenAI technical report.

[18] Babaalla, Z., et al. (2025). LLM-driven MDA pipeline for generating UML class diagrams and code. IEEE Access, 13.

[19] Conrardy, A., & Cabot, J. (2024). From image to UML: First results of image-based UML diagram generation using LLMs. arXiv preprint arXiv:2404.11376.
[20] Nair, R. P., & Thushara, M. (2025). Comparative analysis of large language models for automated use case diagram generation. In Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART) (pp. 401–410).
[21] Ibáñez, M. B., Ruiz-Cristóbal, Á., & Gómez, J. A. (2025). Can multimodal large language models grade like an expert? A study on UML class diagram assessment accuracy. Computer Applications in Engineering Education, 33(5).
[22] Ammersbach, S., Winner, T., & Buxmann, P. (2025). Challenges and feasibility of multimodal LLMs in ER diagram evaluation. Cogent Engineering, 12(1).