Aalto University
School of Electrical
Engineering

Master's Programme in Automation and Electrical Engineering

Understanding the Capabilities of Vision Language Models on UML Sequence Diagrams

Ali Abdul Rehman

Master's Thesis
2025

Aalto University
School of Electrical Engineering

Author Ali Abdul Rehman

Title Understanding the Capabilities of Vision Language Models on UML Sequence Diagrams

Degree programme Automation and Electrical Engineering

Major Controls, Robotics and Autonomous Systems

Supervisor Senior University Lecturer Dr. Jorma Laaksonen

Advisor Msc Laith Zumot

Collaborative partner Nokia

Date 12 October 2025 **Number of pages** 97+12 **Language** English

Abstract

This thesis investigates the capabilities of vision language models (VLMs) to perform visual question answering (VQA) on UML sequence diagrams. For this purpose, a custom dataset was curated from documents, containing annotated sequence diagrams with different levels of visual complexity. The dataset supports three main tasks: information extraction, location recognition, and sequence flow analysis.

The evaluation focused on five important factors: prompt type (zero-shot versus few-shot), context granularity (none, sparse, detailed), the effects of fine-tuning, task type, and diagram complexity. The experiments were carried out on the Gemma family of vision language models, with parameter-efficient fine-tuning applied to the 4B variant using LoRA adapters targeting both the vision and language components. The results indicate that detailed contextual input significantly enhances performance across all types of tasks, while few-shot prompting yields only marginal gains beyond what detailed contextual information alone provides. Fine-tuning led to limited improvements in the current setup, indicating the need for further experimentation with training strategies.

The findings highlight key bottlenecks in applying vision language models to basic perception tasks, including a strong reliance on handcrafted contextual input and variability in performance with different diagram complexities. The thesis concludes by outlining several future research directions, such as full-model fine-tuning, training with longer context lengths, evaluating models with multi-token outputs that involve reasoning, and comparing across a wider range of model architectures and UML diagram types.

Keywords vision language models, visual question answering, UML sequence diagrams, few-shot learning, fine-tuning, model evaluation

Preface

Writing this thesis has been a valuable experience, and I am very grateful to many people who have helped and contributed to helping me reach the finish line.

First, I would like to thank my supervisor, Dr. Jorma Laaksonen, for his constant guidance for the duration of this thesis. He has supported me and given me feedback throughout. I am also indebted to my colleagues at Nokia, particularly my advisor Laith, whose support and advice during our meetings were instrumental in keeping me on track and helping me navigate challenges from start to finish. The culture at Nokia has been immensely supportive, and I truly appreciate the kindness of everyone who offered their thoughts, assistance, and solutions to both minor and major difficulties along the way.

I am very lucky and always grateful to my family and friends for always cheering me on and wishing the best for me. My wife gave me the strength and motivation to complete this thesis, while my parents and sister, despite being far away, have always given unconditional love, support, and encouragement.

Otaniemi, 12 October 2025

Ali Abdul Rehman

Contents

Abstract 3

Preface 4

Contents 5

Abbreviations and Symbols 8

1 Introduction 10
- 1.1 Objectives and Research Questions . . . 11
- 1.2 Structure of the Thesis . . . 12

2 Literature Review 13
- 2.1 Unified Modeling Language and Sequence Diagrams . . . 13
  - 2.1.1 Unified Modeling Language . . . 13
  - 2.1.2 UML Diagrams . . . 13
  - 2.1.3 UML Sequence Diagrams . . . 14
- 2.2 Traditional Methods in Diagram Understanding . . . 18
  - 2.2.1 Rule-Based and Symbolic Techniques . . . 18
  - 2.2.2 Heuristic and Graph-Based Approaches . . . 19
  - 2.2.3 Image Processing and Computer Vision Methods . . . 20
  - 2.2.4 Learning-Based Approaches . . . 26
- 2.3 Large Language Models . . . 27
  - 2.3.1 Transformer Architecture . . . 29
  - 2.3.2 Types of Transformer-Based Models . . . 29
  - 2.3.3 Context and Prompting in LLMs . . . 31
  - 2.3.4 Fine-Tuning LLMs . . . 33
- 2.4 Vision Language Models . . . 35
  - 2.4.1 Architecture of VLMs . . . 36
  - 2.4.2 Pretraining Strategies for VLMs . . . 38
  - 2.4.3 Notable VLMs . . . 39

3 Data 44
- 3.1 Overview of Methodology for Dataset Creation . . . 44
- 3.2 Image Dataset Preparation . . . 46
  - 3.2.1 PDF Parsing and Preprocessing . . . 46
  - 3.2.2 Type-Based Classification of Images . . . 48
  - 3.2.3 Complexity-Based Classification of Images . . . 51
  - 3.2.4 Duplicate Detection and Clustering . . . 55
- 3.3 Textual Context Generation . . . 56
  - 3.3.1 Generation Process and Prompts . . . 56
- 3.4 VQA Question Design and Generation . . . 57
  - 3.4.1 Generation Process and Prompts . . . 58
- 3.5 Final Dataset Summary . . . 60

3.5.1 Training Set Composition 60
3.5.2 Evaluation Set Composition 61

4 Methodology and Experiments **62**
4.1 Research Approach 62
4.2 Experimental Design 63
4.3 Fine-tuning Setup 64
4.3.1 Training Parameters Configuration 65
4.3.2 Computational Environment for Fine-Tuning 66
4.4 Evaluation Prompts 66
4.5 Evaluation Framework 67
4.5.1 Evaluation Strategy 67
4.5.2 Inference Hyperparameters 69
4.5.3 Computational Environment for Evaluation 69

5 Results **70**
5.1 Evaluation Outcomes 70
5.1.1 Role of Context 70
5.1.2 Effect of Prompting Strategies 71
5.1.3 Influence of Fine-Tuning 72
5.1.4 Performance by Task Type 73
5.1.5 Complexity-Wise Performance 75
5.2 Discussion 77
5.2.1 Addressing RQ1: Effect of Context 78
5.2.2 Addressing RQ2: Impact of Few-shot Prompts 79
5.2.3 Addressing RQ3: Effectiveness of Fine-tuning 79
5.2.4 Addressing RQ4: Performance by Task Type 80
5.2.5 Addressing RQ5: Complexity-Wise Trends 80

6 Conclusion **81**
6.1 Key Takeaways 81
6.2 Limitations 82
6.3 Future Direction 82

References **84**

A Representative Examples from the Dataset **98**
A.1 Example of Images 98
A.2 Sparse and Detailed Context Examples 99
A.3 Question/Answer Examples 100

B Prompts Used for Dataset Generation **104**
B.1 Base Instruction for Image Complexity Classification 104
B.2 Prompts for Question/Answer Generation 104
B.2.1 Information Extraction 104
B.2.2 Location Recognition 106

B.2.3 Sequence Flow Analysis 107

C Complete Results **109**

Abbreviations and Symbols

Abbreviations

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
BRIEF	Binary Robust Independent Elementary Features
CAD	Computer-Aided Design
CCA	Connected Component Analysis
CoT	Chain-of-Thought
CRF	Conditional Random Fields
CV	Computer Vision
DoG	Difference of Gaussians
EM	Exact Match
FAST	Features from Accelerated Segment Test
FT	Fine-tune
FT-D	Fine-tuned on *detailed* context
FT-N	Fine-tuned on *no* context
FT-S	Fine-tuned on *sparse* context
HLD	High Level Design
ICC	International Color Consortium
ICL	In-Context Learning
IE	Information Extraction
ITM	Image-text Matching
LLM	Large Language Model
LoG	Laplacian of Gaussian
loRA	Low-Rank Adaption
LR	Location Recognition
MIM	Masked Image Modeling
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
OCR	Optical Character Recognition
PEFT	Parameter Efficient Fine-Tuning
ORB	Oriented FAST and Rotated BRIEF
PDF	Portable Document Format
P&S	Pan and Scan
RGB	Red, Green, and Blue
RoPE	Rotary Positional Embedding
RQ	Research Question
SA	Sequence Flow Analysis
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
TPU	Tensor Processing Unit
UML	Unified Modeling Language
VLM	Vision Language Model
VLSI	Very Large Scale Integration
VQA	Visual Question Answering

Symbols

X	Input sequence embeddings
Q, K, V	Query, Key, Value matrices
W^Q, W^K, W^V	Weight matrices for queries, keys, values
d_k	Dimension of queries/keys
W_1, W_2	Feed-forward network weights
b_1, b_2	Feed-forward network biases
$\sigma(\cdot)$	Activation function
W	Frozen pre-trained weights
ΔW	LoRA weight updates
W_{final}	Combined weights: $W + \Delta W$
d, k	Dimensions of $\Delta W \in \mathbb{R}^{d \times k}$
r	Rank of LoRA decomposition
B, A	LoRA low-rank matrices: $\Delta W = BA$
α	Scaling factor for LoRA updates
Dropout	Regularization probability in LoRA layers

1 Introduction

In the past few years, with the rapid development of artificial intelligence (AI) and Large Language Models (LLMs), Vision Language Models (VLMs) have gained significant attention because of their ability to combine computer vision and natural language processing tasks. The transformer architecture [1] serves as the basis for modern VLMs, which is foundational to the fundamental success of LLMs. VLMs typically consist of two core components: an image encoder that processes visual inputs into embeddings and a text decoder, usually a language model, which generates text based on these embeddings.

In recent years, VLMs have been increasingly explored in different fields such as medicine [2], robotics [3], and literature [4]. Despite their wide range of applications and decent performing capabilities on multiple benchmarks, VLMs, even the best ones, still struggle with basic perception tasks [5]. In particular, they demonstrate difficulty with spatial reasoning, especially when the elements in the images are dense, such as overlapping shapes, the intersection of lines, and multiple arrows.

To improve the performance of models (including vision language models), researchers have developed a couple of methodologies in the past few years, few-shot learning and fine-tuning. Few-shot learning [6] is a learning framework in which a model learns to make accurate predictions by training on a small number of examples during inference, without updating the model parameters. Few-shot is generally referred to as *n-shot learning*, where *n* denotes the number of examples included in the prompt sent to the model during inference. A model inferred at run-time without any examples would be operating in a *zero-shot* setting. On the contrary, fine-tuning refers to the process of adapting and adjusting the parameters of a pre-trained model for specific tasks. This is commonly done by training a model on a smaller dataset that is specific to the use case, allowing it to specialise in that domain. Recent studies show that fine-tuning large language models improves the zero-shot performance of models [7] and for vision language models, fine-tuning is important to align visual features with language generation [8].

Although fine-tuning and few-shot learning enhance the performance of models, the effectiveness and validity of the performance are unreliable due to two major issues: (1) visual content is unnecessary for many samples in the datasets available online, and (2) unintentional data leakage that exists in LLM training [9]. This leads to VLMs often relying on textual cues rather than genuine visual understanding to answer questions. This tendency is further supported by studies that reveal that VLMs strongly prioritize textual information over visual patterns particularly when semantic complexity increases [10]. Other analyses show that VLMs rely heavily on text for simple queries but incorporate visual information as task complexity increases [11]. Another study proposes a text-only training paradigm that improves VLM decision-making without requiring image-text paired data, proving that image data is not strictly necessary for VLMs to achieve strong performances in real-world applications [12].

Despite the growth of work in this field, little research has focused on how few-shot learning and fine-tuning influence a model's dependence on textual context for visual question answering (VQA). While many benchmarks measure overall task performance, they do not distinguish whether improvements arise from better visual understanding or reliance on the language model. As a result, it remains unclear whether fine-tuning or few-shot learning can help VLMs generalize well without external context. If so, prompting with full contextual information at inference will not be necessary. Conversely, if a model still performs poorly afterward and relies heavily on context, it would suggest a significant limitation in current vision language models.

1.1 Objectives and Research Questions

The aim of this thesis is to develop a comparative evaluation framework to assess how fine-tuning and few-shot learning influence the dependence of vision language models on textual context in understanding structured diagrams. For this purpose, the thesis will conduct controlled evaluations of open-source vision-language models under three configurations: zero-shot, few-shot, and fine-tuned. The evaluation will be performed using a private dataset of Unified Modeling Language (UML) sequence diagrams that are not publicly available, thereby eliminating the risk of data leakage. UML diagrams are particularly well-suited for studying basic perception and core visual reasoning tasks. The dataset has been annotated into images of low, medium, and high complexity to ensure the diversity of the data and to prevent models from overfitting to easier ones. The experiments vary the quality of the textual context for an appropriate analysis of the model's dependence on external context. In order to achieve these aims, the thesis focuses on answering the following research questions:

RQ1: How does textual context affect the performance of vision language models when performing VQA on UML sequence diagrams?

RQ2: Does few-shot prompting improve model performance on basic perception tasks compared to zero-shot prompting?

RQ3: Can fine-tuning reduce a model's dependence on textual context, or is high-quality context still necessary?

RQ4: How does the impact of context, prompting, or fine-tuning on model performance vary depending on the type of task?

RQ5: Do models struggle more with a particular complexity of images even when high-quality context is available?

This thesis is limited to the evaluation of a open-source VLMs on a domain-specific dataset of UML diagrams. The selected models were chosen based on their license compliance, organizational integration, compatibility with existing frameworks, and strong overall performance. The focus is to assess how different strategies (zero-shot, few-shot, and fine-tuning) will influence the model's dependence on textual context to achieve meaningful results across three different task types: information extraction, location recognition, and sequence flow analysis. This thesis does not address general benchmarks or public datasets. The dataset used is private and confidential.

1.2 Structure of the Thesis

The remainder of this thesis is organized into five chapters as follows. Chapter 2 provides a comprehensive literature review. It first provides an overview of UML diagrams and sequence diagrams, followed by traditional approaches to diagram understanding before the rise of transformers and LLMs. The next section then introduces the transformer architecture, discusses recent advancements in vision language models as well as a review of context utilization, few-shot learning, and fine-tuning techniques. Chapter 3 presents the data methodology and pre-processing steps taken to create the dataset for the experiments in this thesis. Chapter 4 presents the methodology, including model selection, experimental setup, and configurations used to test the models. Chapter 5 presents and analyzes the results while addressing the research questions. Chapter 6 summarizes the key takeaways, outlines the limitations, and discusses directions for future work.

2 Literature Review

This chapter provides a comprehensive overview of the concepts explained in this thesis. Section 2.1 presents an overview of UML diagrams with a particular focus on sequence diagrams. Section 2.2 discusses traditional approaches to diagram understanding before the emergence of large language models (LLMs) and vision language models (VLMs). Section 2.3 introduces the architecture and functioning of LLMs, while Section 2.4 explores VLMs, which augment language models with visual encoders.

2.1 Unified Modeling Language and Sequence Diagrams

This section introduces the Unified Modeling Language (UML), a standardized visual language to model complex systems. It provides an overview of UML diagrams and their types with a particular emphasis on sequence diagrams. These diagrams represent the primary form of visual data used in this thesis.

2.1.1 Unified Modeling Language

The Unified Modelling Language (UML) is a standard visual representation used to model the structure, behaviour, and interaction of systems [13]. UML emerged from the consolidation of three different object-oriented design methods: the Booch method [14], the Object-Oriented Software Engineering (OOSE) method [15], and the Object Modeling Technique (OMT) [16]. The purpose of this was to standardize the visual representation of software design and methods.

UML has since been widely adopted in various domains, notably in software design, where it has been dubbed the "lingua franca" of the field [17]. Within software engineering, UML has been applied in various subdomains, such as in database design. UML provides a clear visual representation of data structures and relationships. In web application development, it proves a standard approach to model complex architectures [18]. For designing software product lines, UML provides reusable and customizable software components [19]. UML has also proven valuable in architectural-level risk analysis by supporting the visualization and assessment of potential system vulnerabilities, helping software engineering to identify and mitigate such risks [20].

Beyond software engineering, UML's applicability also extends to other fields. UML has been used in business modeling to help visualize organizational structures and processes [21]. In ontology modeling, UML serves as a powerful tool for representing knowledge and relationships between concepts [22].

2.1.2 UML Diagrams

The expressive capabilities of UML are conveyed through its diagrams, which graphically depict the structure and dynamics of a system. These diagrams are composed of graphical elements known as symbols, where nodes represent system elements, and

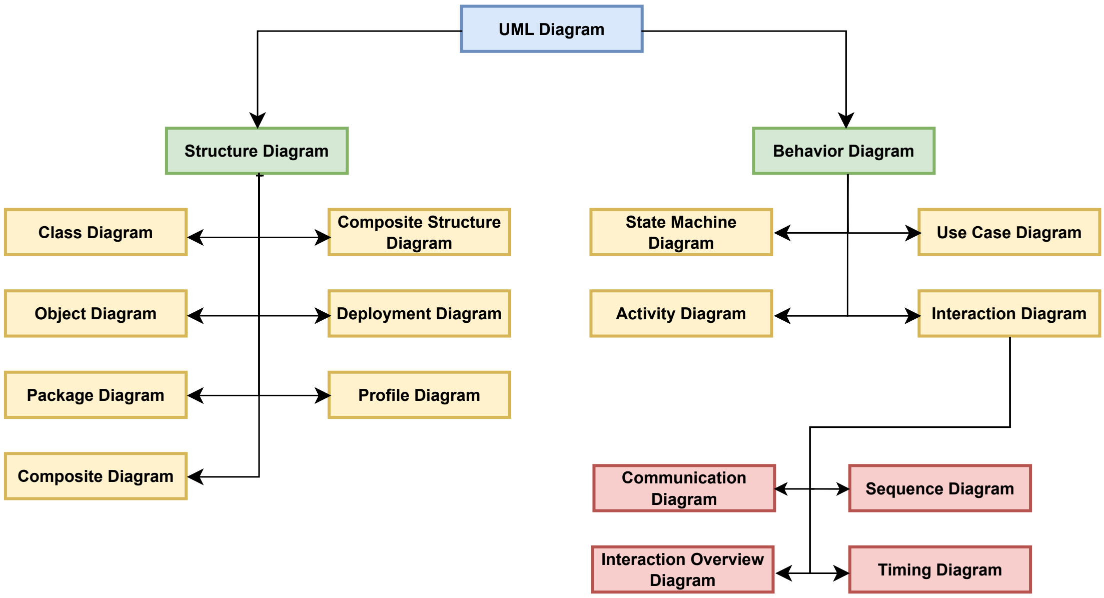


Figure 1: Types of UML diagrams.

edges, also referred to as paths or flows, represent the relationships and connections within a system. The type of UML diagram is defined by the primary graphical symbols it contains. For example, a diagram primarily showing classes is a class diagram, while a diagram that depicts the sequence of message exchanges between different components of a system is called a sequence diagram. UML diagrams are broadly categorized into two main types, structural diagrams and behavioral diagrams. Each category includes multiple subtypes as presented in Figure 1. These two primary categories can be characterized as:

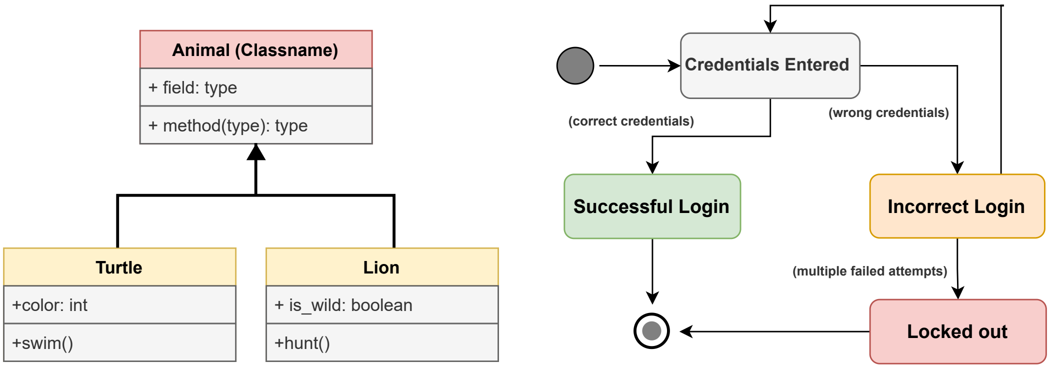
1. **Structural Diagrams:** These diagrams represent the static structure of a system, including its components and the relationships between them. They provide a hierarchical view of the system across various levels of abstraction and implementation. These diagrams capture essential system elements. They do not show concepts related to time.

2. **Behavioral Diagrams:** These diagrams demonstrate the dynamic behavior of objects in a system by focusing on the interaction of components and the change in state over time.

An example of each category is shown in Figure 2.

2.1.3 UML Sequence Diagrams

UML sequence diagrams focus on the exchange of messages between different components of a system over time. They capture the order in which all messages occur, providing a clear view of how various elements of a system interact with each other. By visualizing these message flows sequentially, sequence diagrams make it



(a) Class Diagram (Structural) **(b)** State Machine Diagram (Behavioral)

Figure 2: Examples of structural and behavioral UML diagrams.

easier to understand dynamic processes and system workflows. They also provide an intuitive way to analyze the behaviour of the system. In addition, they are effective in identifying potential bottlenecks, inefficiencies, and errors within a system, thereby supporting system troubleshooting. An example of a UML sequence diagram is shown in Figure 3.

The diagram illustrates the main components through a sample scenario of an ATM cash withdrawal process. Each of these key components is described individually below and shown in Figure 4. The subfigures isolate the major elements in a UML sequence diagram to better understand their roles and interactions.

(A) Lifelines: A lifeline represents an individual participant or object in a sequence diagram. It is placed at the top of the diagram and stretches vertically downward. Lifelines help to follow the sequence of interactions over time for each entity involved. They appear as a rectangular box with a dashed line extending down from the center of the bottom edge.

(B) Actors: Actors represent external entities, such as human users, hardware devices, or other systems, that interact with the system under consideration. They appear at the start of lifelines and usually initiate interactions. The actors show how outside agents communicate with the system through messages or signals.

(C) Activations: Activation bars are thin rectangles placed on lifelines to show the period when an object is actively operating. The top and bottom of the bar match the start and end of the execution.

(D) Messages: Messages represent communication between lifelines and are shown as horizontal arrows. They indicate the flow of information or control between different components of the system. Messages can be broadly classified into the following types:

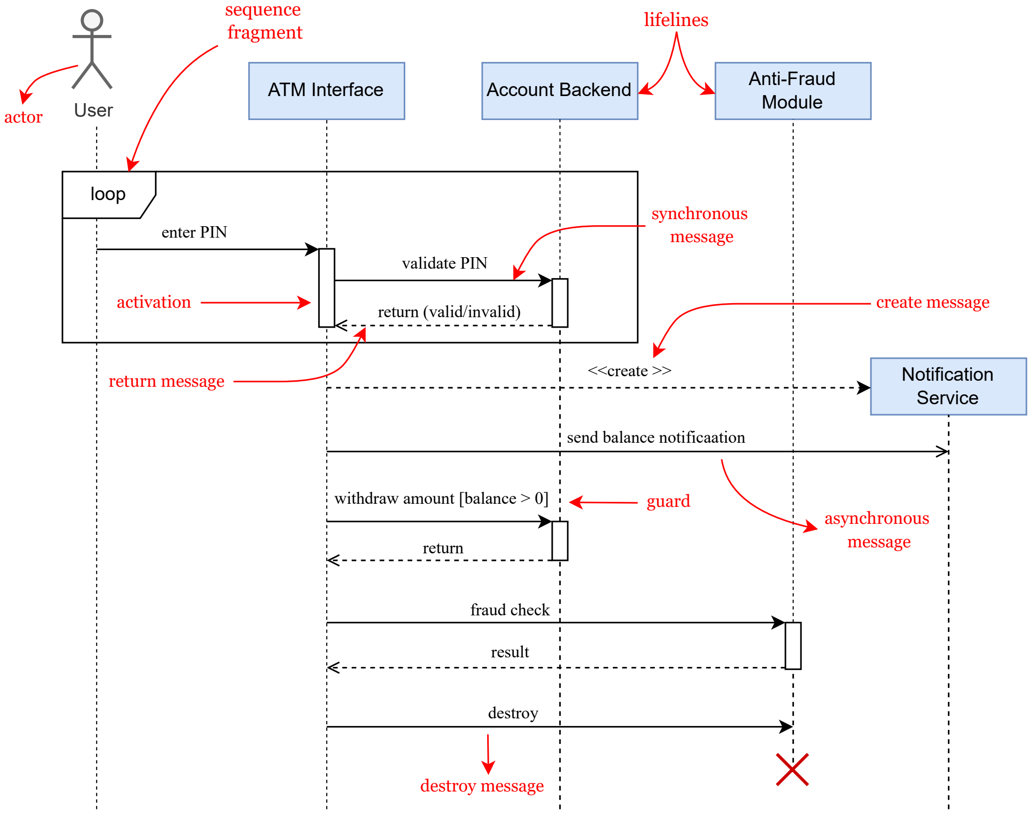


Figure 3: Example of a UML sequence diagram, highlighting the major elements in red

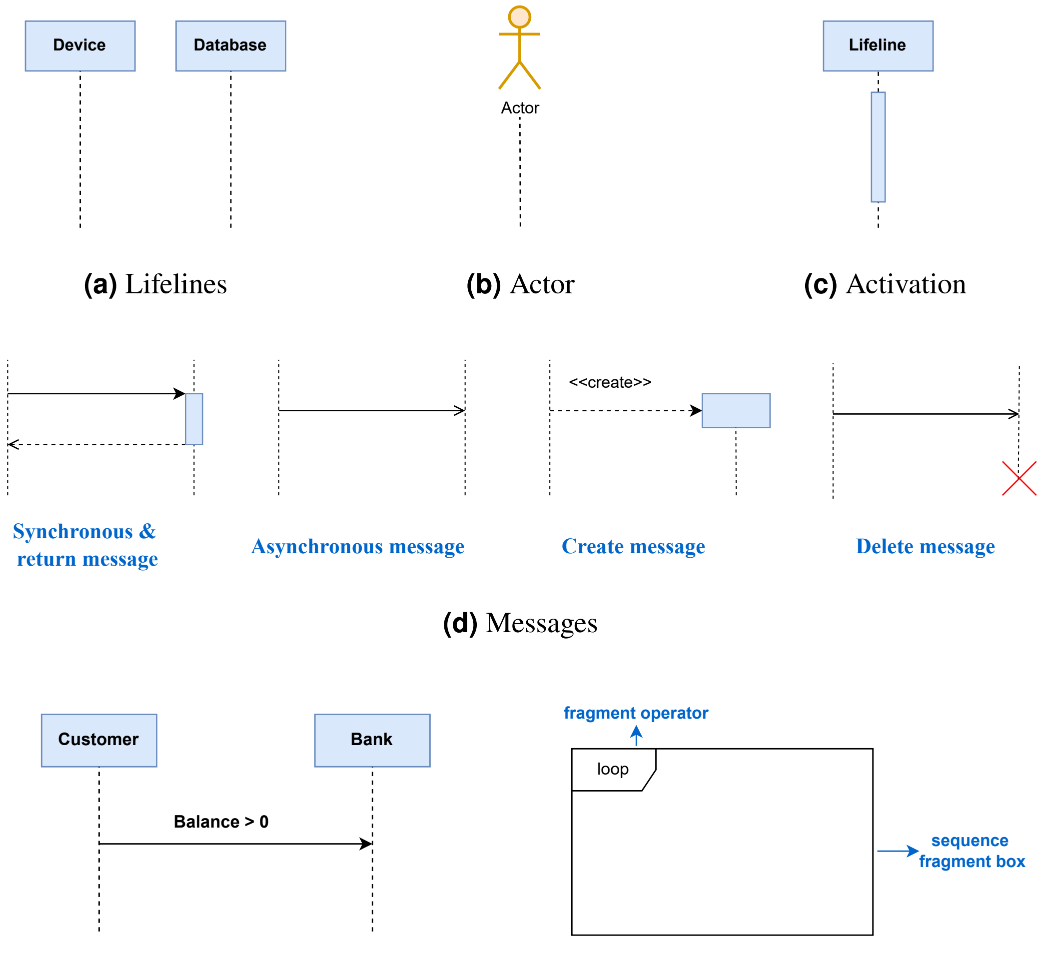
1. **Synchronous message:** A synchronous message requires the sender to wait for a response before proceeding. The interaction pauses until the receiver has finished processing the message and sends a response. It is usually represented as a solid line with a filled arrowhead.

2. **Return message:** A return message shows the response to a synchronous call. It is depicted as a dashed line. This indicates that the control goes back to the sender.

3. **Asynchronous message:** An asynchronous message allows the sender to keep going without waiting for a reply. The interaction continues regardless of whether the receiver has processed the message. It is represented as a solid line with an open arrowhead.

4. **Create Message:** A create message is used to create a new object in the sequence diagram. It appears when a message call results in the creation of an object. This message is shown as a dashed arrow labeled "create."

5. **Destroy/Delete Message:** A destroy message shows that an object in the system is destroyed. It represents the request to destroy the lifecycle of a target lifeline.



(a) Lifelines **(b)** Actor **(c)** Activation

(d) Messages

(e) Guard **(f)** Sequence Fragment

Figure 4: Isolated views of key UML sequence diagram components: (A) Lifelines, (B) Actors, (C) Activation bars, (D) Message types, (E) Guards, and (F) Sequence fragments.

The message appears as an arrow pointing to the lifeline, which ends with an "X" symbol to indicate that the object is destroyed.

(E) Guards: Guards control the flow of messages in a UML sequence diagram. They ensure that a message is sent only if a specific condition is met. For example, the guard condition `[balance > 0]` shows that a cash withdrawal message is sent only when the account has enough funds.

(F) Sequence Fragments: Sequence fragments, also known as combined fragments, are rectangular boxes that enclose parts of a sequence diagram to represent control structures such as conditions, loops, or parallel flows. They help organize

Table 1: Sequence diagram fragment types and their descriptions.

Operator	**Fragment Type**	**Description**
`alt`	Alternative	Only one fragment executes based on the true condition.
`opt`	Optional	Executes only if the condition is true; like a single-branch alt.
`par`	Parallel	Fragments run in parallel.
`loop`	Loop	Repeats while the guard condition is true.
`region`	Critical Region	Only one thread can execute the fragment at a time.
`neg`	Negative	Represents invalid or undesired interaction.
`ref`	Reference	Refers to another interaction diagram; can include parameters.
`sd`	Sequence Diagram	Encloses a complete sequence diagram.

interactions based on different execution paths. The commonly used types of sequence fragments are given in Table 1.

2.2 Traditional Methods in Diagram Understanding

The field of automated diagram understanding has evolved considerably in recent decades, driven by advancements in computer science and artificial intelligence. Researchers and engineers have developed a wide range of computational techniques that enable software systems to interpret and understand diagrams. This chapter provides an overview of the techniques that predate the adoption of large language models (LLMs) and vision language models (VLMs), which are explored in depth later in this thesis.

Section 2.2.1 introduces rule-based techniques based on hand-crafted rules and symbolic reasoning. Section 2.2.3 explores classical computer vision and image processing methods, while Section 2.2.4 explores learning-based computer vision and machine learning techniques that emerged before the advent of language models.

2.2.1 Rule-Based and Symbolic Techniques

Rule-based techniques were among the first approaches used in automating the understanding of diagrams. These methods aim to understand images and diagrams using predefined rules or strategies. They rely on explicitly defined rules, grammars, or templates to comprehend specific types of diagrams. These systems excel in domains with well-defined structures and clear interpretation guidelines, such as circuit diagrams or flowcharts.

In 1986, Niyogi and Srihari [23] introduced a rule-based system for diagram understanding with a top-down goal-directed approach that applied rules on three levels: *knowledge rules* to identify visual features and spatial relations, *control rules* to determine which rules to apply, and *strategy rules* to prioritize and sequence the application of other rules. Another example of rule-based diagram interpretation can be found in the work by Mortensen and Belnap [24], who employed a rule-based approach using feature recognition to characterize engineering graphics. The system using this approach identifies and distinguishes between geometric shapes and features.

A more recent example demonstrates the use of a rule-based framework for recognizing chemical structure diagrams [25]. This approach relies solely on rules to identify bond types, interpret spatial arrangements, and resolve structural ambiguities. Another example is the *evidence rule base* [26], which adds a probabilistic element by assigning weights to visual feature conditions and calculating similarity scores. Although it is probabilistic, the method is based on rules rather than learning-based methods.

2.2.2 Heuristic and Graph-Based Approaches

Heuristic approaches differ from rule-based systems in that they rely on pattern-driven approximations as opposed to hard-coded logic. Heuristic methods are generally more flexible than rule-based ones [27], making them more robust in environments where there may be uncertainty, noise, or variability. However, this flexibility often leads to reduced precision.

Heuristic techniques have been widely used in diagram understanding tasks. Yu et al. [28] proposed a two-stage framework for interpreting diverse engineering diagrams, such as flowcharts, chemical plant layouts, and logic circuits. The system first separates symbols from connecting lines using generalized rules, followed by a domain-specific module that classifies elements through approximate matching. Heuristic rules have also been used to generate UML class diagrams from natural language requirements by applying NLP techniques and rule-based mappings to extract relevant UML elements [29].

Heuristic Reasoning in Early Computer Vision: Many early computer vision methods inherently incorporate heuristic reasoning, forming the foundation for more advanced processing pipelines. Techniques such as thresholding, component analysis, and edge detection rely on carefully crafted rules and assumptions rather than learning from data. For instance, Otsu's method [30] selects a threshold that minimizes intra-class variance, while the Canny edge detector [31] applies Gaussian smoothing and double thresholding to identify edges with high precision. These techniques blend algorithmic design with heuristic assumptions, building a foundation for more advanced processing pipelines, which are reviewed later in this chapter.

Graph-based Methods: Graph-based methods have emerged as an important extension of heuristic techniques for diagram understanding. These approaches utilize the structural properties of diagrams by representing visual elements (symbols, blocks, etc.) as nodes and the relationships between them (arrows, lines, etc.) as edges [32]. This convention is effective in interpreting diagrams that exhibit hierarchical, topological, or network-like structures, including flowcharts, circuit diagrams, and UML diagrams. Graph representations facilitate the integration of multiple possible heuristics, such as proximity constraints and visual similarity metrics.

A notable example is a VLSI-CAD circuit diagram reader that combines heuristics with decision trees and graph-based representations for symbol recognition [33]. A similar example is seen in ECIR, a system for recognizing paper-based electronic circuit diagrams using vectorization and statistical classifiers [34]. A hierarchical object recognition system for circuit diagram analysis was developed using a spatial tree structure and a Bayesian framework [35]. This was later extended with agent-based programming and recursive transformers to improve flexibility and control [36]. In a different domain, a geometric and statistical method has also been proposed to recognize symbols in binary map images [37], utilizing class prototypes and distortion-aware distance metrics.

2.2.3 Image Processing and Computer Vision Methods

Before the emergence of learning-based models, image processing and computer vision techniques were fundamental to automated diagram understanding systems and continue to play a crucial role even today. These methods rely on interpreting diagrams by analyzing their visual structure directly at the pixel level. Instead of relying on past knowledge or training, they use clear algorithmic operations to detect edges, segment regions, extract shapes, and recognize embedded text.

These approaches typically fit into a step-by-step processing pipeline, where each stage processes the image to prepare it for the next steps. This section examines computer vision techniques that are commonly used in interpreting and understanding diagrams.

Image Binarization and Thresholding: Binarization converts grayscale or color images into binary form, assigning each pixel to either foreground (black) or background (white). This process facilitates future steps, such as simplifying vectorization, text and shape recognition, and segmentation, by removing background noise from diagram content. It can be broadly divided into two main categories of thresholding techniques: global thresholding and adaptive thresholding methods.

1. **Global Thresholding:** Global thresholding applies a single intensity threshold to the entire image, separating the foreground from the background. Otsu's method [30] automatically chooses an optimal threshold that maximizes the variance between pixel classes. The minimum error method [38] models the image histogram as a mixture of two Gaussians and selects the threshold that minimizes the classification error. The valley emphasis method [39] changes the objective function of Otsu by focusing on valley points, which are areas of low frequency between intensity peaks in the histogram. A texture-based method iteratively applies Otsu's algorithm and chooses thresholds based on texture features from run-length histograms. This approach works well for complex document images [40]. Global thresholding works well on clear, high-contrast diagrams; however, its performance can be limited for images with poor contrast or background noise.

Table 2: Comparison of global and adaptive thresholding techniques.

Aspect	Global Thresholding	Adaptive Thresholding
Threshold Type	Single threshold applied to the entire image	Varying thresholds computed locally for image regions
Illumination Handling	Works well for uniform lighting conditions	Handles uneven illumination and local contrast variations
Computational Cost	Efficient and fast to compute	More expensive
Common Methods	Otsu, Minimum Error, Valley Emphasis	Niblack, Sauvola, Bernsen, Variational

2. **Adaptive Thresholding:** Adaptive thresholding, also known as local thresholding, addresses the weaknesses of global methods by dynamically adjusting the threshold values for pixels across different regions of the image. Instead of using a single threshold, it calculates local thresholds, allowing it to adapt to variations in illumination, contrast, and texture. This approach is particularly effective in handling non-uniform lighting conditions and complex textures while preserving fine details and edges that might be lost with global methods. A well-known technique is Niblack's method [41], which computes pixel-wise thresholds using the local mean and standard deviation. Sauvola improved upon Niblack's method by introducing a dynamic factor that adjusts the threshold based on local contrast [42], improving performance for regions with varying contrast levels. Beyond these methods, many other adaptive approaches have been proposed. Bernsen's method uses local contrast within a fixed window, making it effective in noisy or low-contrast areas [43]. A variational approach minimizes an energy function to compute smooth threshold surfaces for uneven backgrounds [44]. A tangent flow-based method enhances edges while preserving key visual details in binary form. These are only a few examples among a wider variety of adaptive thresholding methods discussed.

Table 2 provides a comparison of the key differences between the global and adaptive thresholding methods discussed.

Edge Detection: Edges in an image are characterized by sharp changes in pixel intensity, often marking the boundaries of objects. Edge detection is the process of identifying the regions with sharp intensity transitions, which, in diagrams, correspond to the edges of various elements.

A common way to detect these transitions is by using gradient-based methods, which compute the first derivative of the image intensity to reveal changes in a given direction. High-gradient magnitudes typically indicate edges. Several classical edge detection operators are based on gradient estimation, which are given below:

1. **Sobel operator:** The Sobel operator [45] computes the first derivative in

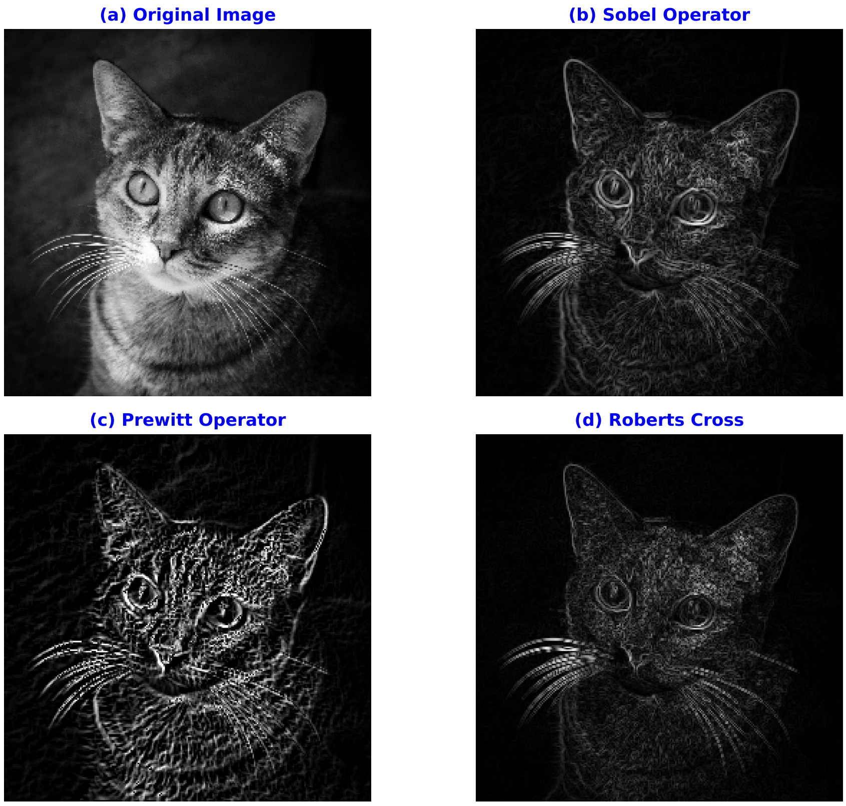


Figure 5: Edge detection results using classical gradient-based methods: (a) Original image, (b) Sobel, (c) Prewitt, and (d) Roberts cross operator.

horizontal and vertical directions using convolutional filters, combining edge detection with noise smoothing.

2. **Prewitt operator:** Similar to the Sobel operator, the Prewitt operator [46] uses uniform weights in its convolutional kernels. While it is computationally simpler, it is slightly less accurate in detecting diagonal edges.

3. **Roberts cross operator:** The Roberts cross operator [47] applies a kernel 2×2 to detect edges along diagonals with high sensitivity. However, it is more susceptible to noise due to its small size.

Figure 5 depicts the output of each of these techniques when applied to the same input image.

Another class of edge detection techniques uses second-order derivatives and identify edges by detecting zero-crossings in image intensity. Methods like the Laplacian of Gaussian (LoG) and the Difference of Gaussians (DoG) [48] are more sensitive to fine details and are effective for detecting thin or faint edges. LoG smooths the image with a Gaussian filter before applying the Laplacian, while DoG approximates this by subtracting two Gaussian-blurred images with different standard deviations. In addition to the gradient and second-order derivative techniques, the

Canny edge detector [31] represents a more advanced approach. It combines several steps, including Gaussian smoothing, gradient calculation, non-maximum suppression, and double thresholding, to optimize accurate edge detection.

Morphological Operations: After thresholding or edge detection, the initial binary image may have imperfections, such as lines with gaps, random specks of noise, or varying line thickness. Morphological operations are techniques used to analyze and modify the shapes and structures of objects within images [49]. These techniques mainly cater to binary images, but can also be used for grayscale images. Common morphological operations used in image and diagram preprocessing [50] are as follows:

1. **Erosion** removes pixels along the boundaries of objects to remove noise and separate connected objects or boundaries.
2. **Dilation** adds pixels to expand shapes and connect nearby elements. It is the opposite of erosion.
3. **Opening** consists of erosion followed by dilation to remove small objects and noise while preserving the shape of larger structures.
4. **Closing** combines dilation followed by erosion to close small holes and gaps in objects.
5. **Morphological Gradient** first applies erosion and dilation separately on the image. Then, it calculates the difference between the eroded image and the dilated image.
6. **Top-Hat Transform** extracts small elements and details from images.
7. **Hit-or-Miss Transform** detects patterns and configurations in binary images.
8. **Skeletonization** reduces objects to their minimal one-pixel-wide shape while preserving topological structure and connectivity.

Connected Component Analysis and Contour Detection: In diagram understanding, segmenting visual elements is essential for identifying symbols, text blocks, and other structural components. Connected Component Analysis (CCA) is a labeling technique that groups adjacent foreground pixels into separate components based on connectivity [51]. Each connected region can be examined independently for properties such as size, aspect ratio, centroid location, or bounding box. This technique is helpful for segmenting elements in diagrams, especially after cleaning the binary image with morphological operations.

Contour detection outlines the precise boundaries of connected components by tracing their outer edges at the pixel level [52]. This technique captures the exact shape of each element. Instead of treating connected areas as single blobs, it captures the ordered outlines of objects, providing detailed geometric information. Although contour detection and edge detection are closely related, they have different roles in

Table 3: Comparison between edge detection and contour detection.

Aspect	Edge Detection	Contour Detection
Robustness	Sensitive to noise and may produce false edges	Robust to noise; provides reliable boundary representations
Image Type	Applied on grayscale images	Applied on binary or preprocessed grayscale images
Purpose	Highlights regions of intensity change	Returns ordered boundary points of connected boundary points
Connectivity	Edges may be broken or fragmented	Contours are closed, connected loops

image analysis. Edge detection highlights areas where there are quick changes in intensity. This results in a binary edge map that marks possible boundary points for objects. Contour detection works on binary images, which often come from edge maps or thresholding, and extracts complete boundary outlines as ordered sequences of coordinates. The key differences between edge detection and contour detection are presented in Table 3.

Feature Detection and Matching: The next step in diagram understanding is to identify unique points or patterns that represent key structural or symbolic elements. In computer vision, such points or patterns are called features [53]. Feature detection algorithms identify areas that are visually *interesting* and can act as reliable anchors for tasks like recognition or matching. Although there is no universally accepted definition of a feature, it is generally understood to be a salient part of an image, such as a corner or edge which stands out due to local contrast or geometric structure. A good feature detector should demonstrate repeatability [54]. This means that it can consistently identify the same features in different images of the same scene. The three common types of image features are:

1. **Edges**, which represent points where there is a boundary between two regions of the image.

2. **Corners or interest points**, which refer to point-like features with a local two-dimensional structure. These are points where the intensity changes significantly in multiple directions.

3. **Blobs or regions of interest**, which are uniform regions that differ from their surroundings in texture or brightness. Blob detectors are useful for identifying smooth or rounded areas that lack clear edges or corners.

4. **Ridges**, which are one-dimensional curves that trace the central axes of elongated structures. They capture symmetry and local width but are harder to detect than edges, corners, or blobs.

Once the types of features are identified, the next step is to choose suitable algorithms to detect them. Among the first and most influential methods is the Harris Corner Detector [55]. This detector identifies interest points by assessing how image intensities change in all directions within a small window. Although the Harris corner detector is efficient, it lacks scale invariance, which limits its effectiveness in scenarios involving images at varying resolutions. To overcome this limitation, the patented Scale-Invariant Feature Transform (SIFT) algorithm [56] was proposed. It detects keypoints at various scales using the difference of gaussians (DoG) method and computes gradient-based descriptors around each point. SIFT features are robust to changes in scale, rotation, and moderate affine transformations. This makes them perfect for matching diagram elements taken at different zoom levels or orientations. However, SIFT was not fast enough for many practical applications. Speeded-Up Robust Features (SURF) [57] builds on SIFT by using box filters and integral images to accelerate detection. SURF also introduces strong descriptors based on Haar wavelet responses. A comparative study evaluated the performance of SIFT and SURF under various object and background conditions [58]. The findings indicate that SIFT can outperform SURF in situations involving image blur and illumination changes. However, SURF always runs faster.

Although SIFT and SURF perform well, their computational demands make them less suitable for real-time applications such as Simultaneous Localization and Mapping (SLAM). To address this, the FAST (Features from Accelerated Segment Test) algorithm [59] was introduced as a high-speed corner detection method. FAST finds corners by looking at the intensity of a circular ring of pixels around a candidate point, allowing for quick processing. FAST detects corners by looking at the intensity values of a circular ring of 16 pixels around a candidate point. If a group of adjacent pixels in this ring is much brighter or darker than the candidate pixel, the point is marked as a keypoint. To improve processing speed, a quick test is first applied to a subset of four pixels. A complete segment test is only done if the point passes this initial check. The complete procedure is summarized in Algorithm 1.

However, FAST lacks scale invariance and does not provide feature descriptors. BRIEF (Binary Robust Independent Elementary Features) [60] is a lightweight feature descriptor that works with any keypoint detector, including FAST. It generates compact binary strings by comparing the intensity values of predefined pairs of pixels in a local patch around each keypoint. It is a fast method for computing and matching feature descriptors, offering high recognition rates except in the case of significant in-plane rotation.

The Oriented FAST and Rotated BRIEF (ORB) algorithm [61] merges the FAST keypoint detector with a rotation-aware version of the BRIEF descriptor. This combination addresses the weaknesses of both methods while providing a computationally efficient, patent-free alternative to SIFT and SURF. ORB begins by detecting keypoints using FAST and ranks them with the Harris corner measure to retain the most informative ones. It uses image pyramids for scale invariance. Since FAST does not compute orientation, ORB estimates keypoint orientation by calculating the intensity-weighted centroid of each patch, allowing for rotation-invariant descriptors. It improves its effectiveness by selecting binary tests with high variance and low correlation, leading

Algorithm 1 FAST Corner Detection Algorithm

Require: Image I, threshold t

for all pixels p in I **do**
    $I_p \leftarrow$ intensity at p
    Select 16 pixels in a circle around p
    if at least 12 contiguous pixels are all $> I_p + t$ or all $< I_p - t$ **then**
        Perform high-speed test at positions 1, 5, 9, and 13
        if at least 3 of the 4 pixels pass the threshold **then**
            Apply full segment test
            if segment test passed **then**
                return p as a keypoint
            end if
        end if
    end if
end for

to a refined descriptor called rBRIEF. Feature matching is accelerated using multiprobe Locality Sensitive Hammering (LSH) [62], making ORB well suited for real-time applications. The practical flow of the ORB algorithm is shown in Figure 6.

2.2.4 Learning-Based Approaches

With the increase in labeled data and increased computational power, machine learning (ML) techniques have emerged as a strong alternative to manually defined rules and heuristics for understanding diagrams. These methods allow systems to learn patterns and features from data instead of depending on fixed logic. This makes them more flexible in handling variations in diagram styles, noise, and incomplete or unclear inputs.

Classic Machine Learning Methods: Early learning-based methods focused on extracting hand-engineered features from diagram images. These features included shape descriptors, texture patterns, geometric relations, and topological attributes. They were then fed into traditional classifiers such as Support Vector Machines (SVM) [63], Random Forests [64], k-Nearest Neighbours (k-NNs) [65], and Naive Bayes [66] classifiers. These models were trained to classify individual components, such as symbols, connectors, and text blocks, or entire diagrams, into specific categories. Some practical applications of these methods include offline recognition of hand-drawn

Figure 6: Workflow of the ORB algorithm.

structured diagrams using SVMs [67], digital logic circuit component recognition using SVMs [68], and handwritten character classification using Random Forests [69].

Graphical and Probabilistic Models: Graphical models are widely used to capture relationships among diagram elements [70]. Diagrams can be viewed as graphs, where nodes represent symbols or text blocks, and edges show visual connectors like lines or arrows. Probabilistic Graphical Models (PGMs), including Conditional Random Fields (CRFs) [71] and Hidden Markov Models (HMMs) [72], provide a framework for labeling these structures. For instance, a Bayesian CRF framework has been used to recognize hand-drawn diagrams. The classification of each element depends on the labels and features of its neighbors, making the system more robust against local ambiguities [73].

Deep Learning Methods: Deep learning has greatly improved image understanding by eliminating the need for manually created features. Convolutional Neural Networks (CNNs) have become the main tool for classifying image-based diagrams [74]. These models learn spatial hierarchies and visual patterns directly from pixel data. This ability makes them very effective at identifying symbols, structures, and layouts in diagrams.

In the context of UML diagram analysis, deep learning models have shown strong performance across various tasks. One study proposed a CNN-based tool for automatically classifying UML class diagrams from other diagram types, accepting raw image files as input and requiring no manual annotation [75]. Beyond classification, deep learning has also been applied to more specialized problems such as detecting design patterns in UML class diagrams [76] and generating UML activity diagrams from business process text using semantic role labeling and graph modeling techniques [77].

Comparison with Computer Vision Techniques: Despite their success, learning-based methods, especially deep learning, require large labeled datasets and often operate as black-box systems with limited transparency into how predictions are made. In contrast to traditional computer vision methods, which depend on algorithmic tasks, deep models focus on learning features from data. CV methods are typically fast, easy to understand, and work well with clean, organized diagrams. However, they have difficulty handling changes in layout, noise, or unclear inputs. Deep learning methods can be generalized to different types and styles of diagrams, but they require more computational power and are less transparent. Table 4 summarizes the differences between traditional computer vision techniques and deep learning based approaches.

2.3 Large Language Models

Large Language Models (LLMs) are neural networks trained on vast amounts of data [78]. They are capable of understanding, generating, and reasoning with human language. LLMs predict the next word or token in a sequence, enabling them to

Table 4: Comparison of traditional computer vision techniques and deep learning-based approaches.

Aspect	**Computer Vision**	**Deep Learning**
Feature Engineering	Hand-crafted features designed by experts	Automated feature learning from data
Data Requirements	Performs well with limited data	Requires large datasets for training
Computational Resources	Generally less demanding, suitable for real-time applications	High computational power needed, especially during training
Performance	Effective for simpler tasks	Superior performance on complex tasks
Interpretability	More interpretable due to explicit feature extraction and algorithmic steps	Often considered a "black box" due to abstract feature representation
Flexibility	Limited flexibility, often tailored for specific tasks	High flexibility, can generalize to various tasks
Development Time	Requires significant manual effort for feature engineering	Longer training time but less manual intervention for feature extraction
Adaptability	Less adaptable to new tasks or changes in the environment	Highly adaptable, can learn new tasks with additional training data
Real-Time Capability	Good for real-time processing on limited hardware	Real-time processing possible but requires powerful hardware

perform a wide range of tasks. LLMs are typically built on the transformer architecture and contain billions of parameters, which allows them to recognize complex patterns in language. LLMs have become a major breakthrough in natural language processing, driving significant improvements in tasks such as text generation, summarization, translation, and question answering.

LLMs serve as the language backbone for Vision-Language Models (VLMs), which build upon their capabilities by incorporating visual input processing. These VLMs form the central focus of this thesis. This section introduces the primary design principles that underlie LLM, with a focus on transformer architecture. It also explores different types of models derived from transformers, including encoder-only, decoder-only, and encoder-decoder variants.

2.3.1 Transformer Architecture

The transformer architecture, introduced in the 2017 paper *Attention Is All You Need* [1], serves as the foundational building block of modern LLMs. Transformers changed natural language processing by replacing recurrence with self-attention, a mechanism that allows models to evaluate the significance of various parts of an input sequence while encoding a specific element of that sequence. This change enables models to process entire sequences simultaneously. As a result, transformers can better capture long-range dependencies and contextual relationships than earlier structures, such as recurrent neural networks (RNNs) [79, 80].

The transformer consists of stacked layers, each containing a multi-head self-attention mechanism followed by a position-wise feed-forward network, where self-attention is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

with $Q = XW^Q$, $K = XW^K$, and $V = XW^V$. Specifically, in the attention mechanism, this includes the query, key, value, and output projection layers that compute attention scores and transform the attended features. The feed-forward network is given by

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2.$$

The multi-head self-attention enables the model to examine information from multiple representation subspaces simultaneously, while the feed-forward network refines the representation of each token independently. To keep track of the order of tokens, which the model does not naturally encode, positional encodings are added to the input embeddings at the base of the architecture. These encodings enable the model to distinguish between positions in a sequence.

A simplified version of the original transformer architecture is illustrated in Figure 7. The original transformer design consists of an encoder and decoder architecture. The encoder processes the input data and produces a set of contextual embeddings that capture the relationships between tokens. The decoder generates the output sequence by looking at its previously generated tokens and the representations from the encoder. The decoder uses masked self-attention, which prevents it from accessing future tokens during training. To prevent information leakage during training, the decoder uses masked self-attention, which prevents it from accessing future tokens during training.

The parallelizable nature of self-attention allows significantly faster training [81]. Instead of processing one token at a time in sequence, transformers connect all positions in an input simultaneously using a fixed number of sequential operations. These operations can be spread out across tokens in parallel, making transformers highly scalable and ideal for large-scale training.

2.3.2 Types of Transformer-Based Models

Although the original transformer architecture includes both an encoder and a decoder, many modern large language models use only part of this design. They adapt it to meet

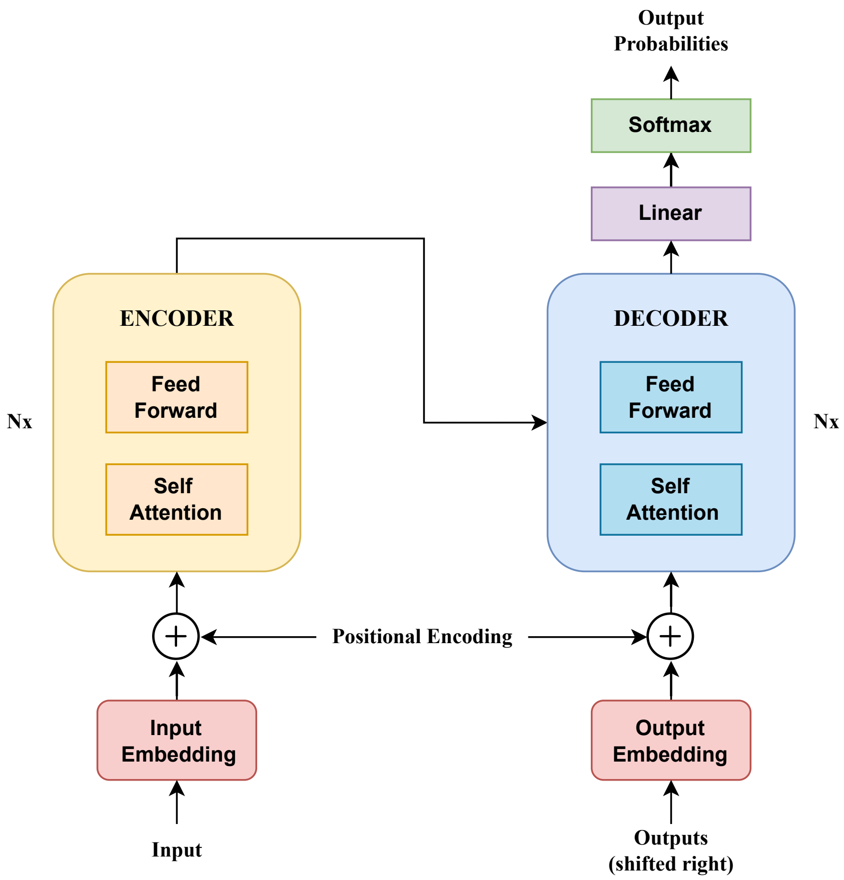


Figure 7: Simplified version of the original transformer architecture.

the needs of different use cases. Using either the encoder or decoder alone, or both together, researchers have created more efficient, scalable, and task-specific models. These models can be broadly categorized into three types based on their architecture:

1. Encoder-only models
2. Decoder-only models
3. Encoder-decoder models

The following paragraphs provide details of each of these model types.

Encoder-only Models: Encoder-only models consist only of the encoder stack from the original transformer architecture. Encoder models process input sequences bidirectionally, forward and backward. This allows them to capture the full context of each token. These models are small and fast. A prominent example of an encoder-only model is BERT (Bidirectional Encoder Representations from Transformers) [82]. BERT is pretrained on a large text corpus with two self-supervised goals: masked language modeling (MLM) and next sentence prediction (NSP). In masked language modeling, the model masks random tokens in the input and learns to predict them based on the surrounding context. The next sentence prediction trains the model to figure out if two input sentences follow each other in the original text. These pretraining tasks help BERT learn deep contextual representations, which can be fine-tuned for tasks such as sentiment analysis and question answering.

BERT remains one of the most widely used and influential language models in natural language processing. Its design and architecture have formed the basis for many subsequent models, such as RoBERTa (Robustly Optimized BERT Pretraining Approach) [83], which improves on BERT by removing the next sentence prediction objective, training on a larger dataset, and using dynamic masking.

Decoder-only Models: Decoder-only models retain only the decoder stack of the original transformer architecture. They work in an autoregressive fashion. This means that the model generates one token at a time, conditioning each prediction on all previously generated tokens [84]. This is achieved through causal (left-to-right) self-attention, which ensures that the model can only attend to earlier positions in the sequence, preventing it from accessing future tokens during training and inference. Due to this sequential token prediction mechanism, decoder-only transformers are highly effective for generative tasks such as text completion and summarization [85]. A prominent example of this architecture is the GPT (Generative Pre-Trained Transformer) family. [86].

Encoder-decoder Models: The encoder-decoder models preserve the full structure of the original transformer, with an encoder and a decoder stack. The encoder processes the input sequence to create a detailed contextual representation, while the decoder uses this representation through cross-attention to produce the output sequence. These models have separate input and output stages. This separation improves performance on tasks that require a robust connection between the source and target sequences, such as sequence-to-sequence tasks, such as machine translation [87]. Well-known examples include T5 (Text-to-Text Transfer Transformer) [88] and BART (Bidirectional and Auto-Regressive Transformers) [89].

2.3.3 Context and Prompting in LLMs

LLMs operate as conditional probability estimators that predict the next token based on a contextual window of preceding tokens. This context acts like the model's short-term memory and helps the model in understanding instructions, keeping coherence, and handling different tasks without needing to retrain. The context defines what the model knows during a single inference session. To make an LLM perform a task, an LLM relies on natural language instructions, known as prompts [90]. In-context learning (ICL) refers to the ability of language models to infer and perform new tasks by observing instructions and examples given in their input context, rather than through additional training [91].

Table 5: Example of zero-shot and few-shot prompting in LLMs.

Prompting Type	Example
Zero-Shot Prompting	*Prompt:* "Translate 'Good morning' into French." *Output:* "Bonjour."

Few-Shot Prompting	*Prompt:*
	"`English → French`
	`Hello → Bonjour`
	`Thank you → Merci`
	`Good night → Bonne nuit`
	`Good morning → ?`"
	Output: "Bonjour."

Modern LLMs are often seen as few-shot learners [6], meaning they can understand new tasks after being shown just a small number of labeled examples within the prompt itself. This differs from zero-shot learning, where the model is given only a natural language instruction without examples and must rely entirely on its pretraining knowledge. Both capabilities arise from the ability of the transformer architecture to recognize patterns and relationships in long sequences of text. Table 5 shows an example of zero-shot and few-shot prompting.

Recent advancements in context window size have allowed models to work in the many-shot regime. In this mode, users can provide hundreds or even thousands of examples in the prompt. Studies show that many-shot prompting can greatly improve model performance in various reasoning and classification tasks [92]. A prompting technique known as chain-of-thought (CoT) prompting has been proposed to further improve reasoning in few-shot settings [93]. CoT works by including intermediate reasoning steps in the prompt examples. This allows the model to break down complex tasks into smaller, understandable steps. This approach has been shown to significantly improve performance on various tasks, especially when applied to larger models, which are better at reasoning.

Prompt engineering is the practice of carefully designing and structuring input prompts to effectively influence the behavior of large language models [94]. Since LLMs generate output entirely based on the context provided in their input window, even minor adjustments in wording, format, or adding examples can greatly affect the quality and accuracy of the results. Well-crafted prompts can improve coherence, reduce hallucinations, and align the model's output with the desired format.

Recent studies have shown that prompt design and formatting choices can greatly impact LLM performance, even when the underlying input content remains the same [95]. Similarly, an LLM's ability to recall specific information in a prompt, which is a key part of ICL, varies by model and by the structure of the prompt [96]. Recall accuracy depends on the position and phrasing of the information within the prompt, highlighting the importance of prompt engineering. Additionally, LLMs can generate output that contradicts the input context, particularly when memorized knowledge overrides the context provided for the task [97]. This phenomenon, known as contextual unfaithfulness, poses challenges in tasks like knowledge extraction and reading comprehension. However, prompt engineering techniques such as opinion framing and counterfactual demonstrations have been shown to improve alignment with the given context without requiring additional training.

2.3.4 Fine-Tuning LLMs

Although prompting allows LLMs to adjust to new tasks without changing model parameters, there are many situations where in-context learning alone is not enough, particularly when dealing with specialized fields or tasks that require high precision. In such cases, fine-tuning is used to adapt models for specific downstream tasks by updating their internal parameters using labeled examples [98].

Fine-tuning begins with the selection of a pre-trained model that has been trained on a large and diverse set of data. This model is then adapted to a specific downstream task using a smaller, task-specific dataset. The early layers of the model are often kept frozen to reduce computational costs and to keep the general representations learned during pretraining. The later layers, which capture more specific features for the task, are updated using the new training data. This method allows the model to fit the target task while keeping the stability of its pre-trained weights. After training is finished, the model is tested on the task.

In scenarios where computational resources are limited or full fine-tuning is impractical, parameter-efficient fine-tuning (PEFT) methods offer an effective alternative [99]. These methods aim to adjust large-scale language models for new tasks without updating all the model's parameters. Instead of retraining the entire network, PEFT techniques modify only a small number of task-specific parameters while keeping most of the model unchanged. This approach greatly reduces computational costs, memory use, and the risk of overfitting. At the same time, it maintains strong performance in various applications [100]. PEFT methods can outperform in-context learning in terms of accuracy and efficiency, particularly by avoiding the need to reprocess task examples during inference [101].

Low-Rank Adaptation: One widely adopted PEFT technique is LoRA (Low-Rank Adaptation), which introduces trainable low-rank matrices into the attention layers of a transformer while keeping the original model weights frozen [102]. During training, LoRA freezes all the pre-trained weights W and instead learns a set of low-rank update weights ΔW that encode task-specific adaptations. At inference time, these updates are added back to the frozen weights via simple addition, $W_{\text{final}} = W + \Delta W$, enabling the model to perform the desired downstream task.

Figure 8 shows the difference between regular fine-tuning and LoRA-based adaptation. Assuming ΔW is a full-rank matrix of size $d \times k$, LoRA reduces redundancy through low-rank decomposition, rewriting it as $\Delta W = BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times d}$, with $r \ll d$ being a tunable rank parameter. The rank r controls the inner dimension of the low-rank matrices, where higher values allow for more expressive weight updates but increase memory and computational costs. In addition, α is a scaling factor applied to the LoRA updates before they are added to the frozen model

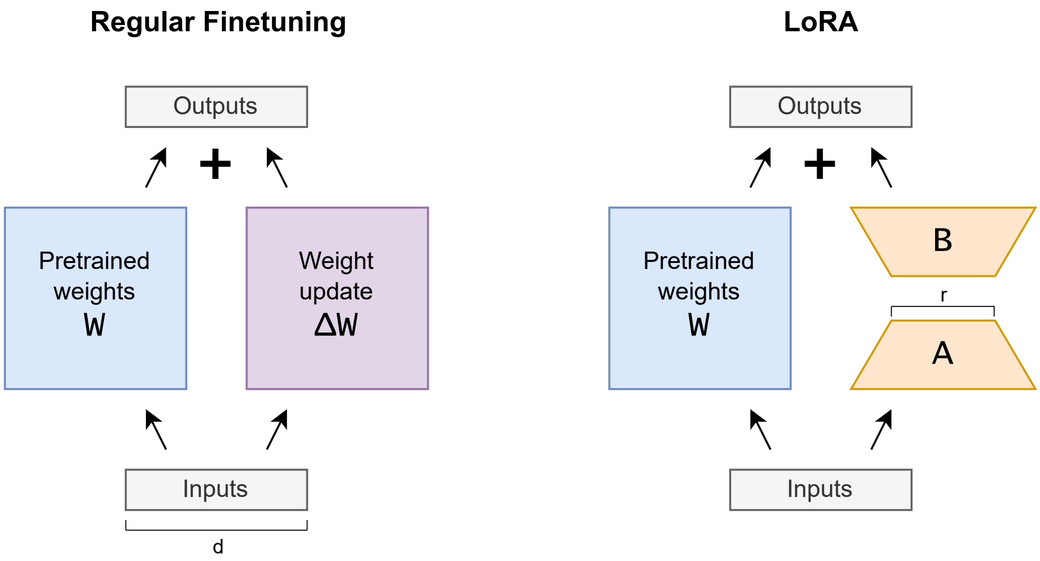


Figure 8: Comparison between regular fine-tuning and LoRA. LoRA learns low-rank matrices A and B to approximate the weight update ΔW, while keeping the pretrained weights W fixed.

weights, expressed as $\Delta W = \alpha BA$, modulating the magnitude of learned adaptations and providing a balance between stability and learning speed. Dropout can also be applied. It indicates the likelihood of randomly zeroing elements within the LoRA layers during training. This serves as a form of regularization, which helps reduce overfitting and enhances generalization.

Instead of directly training the full matrix ΔW, LoRA trains the smaller matrices B and A, drastically reducing the number of trainable parameters while retaining performance. For example, a 3×3 matrix has nine parameters; choosing $r = 1$ results in only six trainable parameters $(3 + 3)$, while still preserving most of the relevant information. LoRA can be applied to all projection layers in the multihead attention and feedforward (MLP) blocks of the language model, as well as to the corresponding projection layers in the vision tower encoder. In the attention mechanism, this includes the query, key, value, and output projection layers. In MLP blocks, the target layers are the up-projection, down-projection, and gating layers, which control the transformation and scaling of intermediate hidden states.

When using LoRA, only the adapter parameters are updated during training, while the base model weights remain unchanged. Consequently, saving the model after training only preserves the adapter weights. For deployment, these adapter weights can be merged into the base model weights, eliminating the need for separate LoRA modules during inference and ensuring compatibility with the evaluation framework.

Although LoRA significantly reduces training time and computing requirements, its limitations on parameters can sometimes restrict how well it adapts to specific tasks. As a result, while it often achieves competitive performance, the final results

Table 6: Comparison of in-context learning vs. fine-tuning.

Criterion	In-Context Learning (ICL)	Fine-Tuning
Knowledge integration	Uses instructions and examples directly in the prompt; model weights remain unchanged	Updates model parameters to permanently incorporate task-specific knowledge
Data	Works well with few examples	Requires larger datasets
Deployment time	Immediate deployment; no retraining required	Requires additional training before deployment
Cost and resources	Lower upfront cost; higher inference cost due to long context windows	Higher training cost; lower inference cost once fine-tuned
Flexibility	Easily adaptable to multiple tasks without retraining	Optimized for a single or narrow set of tasks

may not always reach the level of fully fine-tuned models, particularly for specialized or complex tasks.

Comparison between ICL and Fine-Tuning: While in-context learning and fine-tuning are both strategies for adapting large language models to new tasks, they differ in how knowledge is incorporated. In-context learning adapts the behavior of the model by providing task instructions and examples directly in the input context, without changing the model parameters. Fine-tuning updates some or all of the model's weights to permanently include task-specific knowledge.

Studies comparing the two approaches have found that the advantages of the two methods depend on the nature of the task, the size of the model, and the number of available examples [103]. For tasks with implicit patterns, in-context learning captures patterns more effectively [104], while fine-tuning can perform better than ICL when there are large, high-quality datasets and deeper parameter adjustments are needed. Both methods can generalize well under certain conditions, and neither consistently outperforms the other across all task types.

In practice, selecting between ICL and fine-tuning means understanding the specific use case and considering the trade-offs in performance, cost, and how well each method fits the task. Fine-tuning can produce compact, highly specialized models but demands additional training time and careful preparation of the dataset. ICL may require a larger model to achieve similar results, but it can still work effectively with less data. The key differences between the two approaches are summarized in Table 6.

2.4 Vision Language Models

Vision Language Models (VLMs) extend the capabilities of large language models by including visual inputs with textual data [105]. The basic concept of VLMs is to create a joint representation of visual and textual data in a single embedding space. When an image and optional text are given as input, VLMs produce textual outputs.

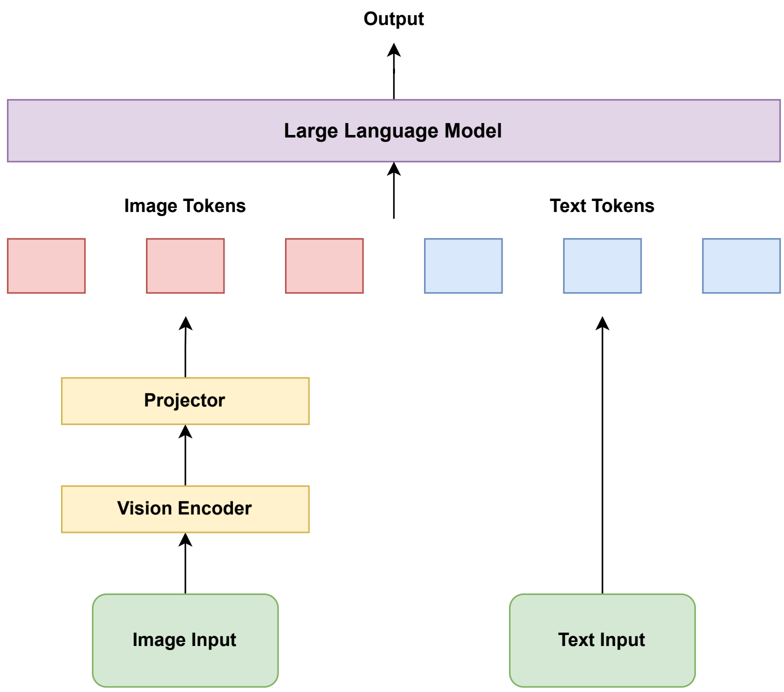


Figure 9: General architecture of a vision language model.

This allows them to be used in multiple use-cases such as image captioning [106], document understanding [107], and visual question answering (VQA) [108].

There is a great diversity within the existing set of VLMs [109]. This section first outlines the general architecture of VLMs and then highlights the specific models relevant to this thesis. In the end, there is an overview of the current state of VQA within VLMs.

2.4.1 Architecture of VLMs

VLMs have different architectural designs, but generally consist of three main components [110]:

1. Vision module.
2. Language module.
3. Fusion mechanism.

A typical VLM combines these three components to enable multimodal joint processing, as shown in Figure 9. The following paragraphs describe each of these components in detail.

Vision Module: The vision module is responsible for transforming raw visual input, such as images and diagrams, into a structured representation to fuse with the language module. It typically comprises the following:

1. **Vision Encoder:** A vision encoder processes image input to extract essential visual features, such as color, shape, and texture. Then it transforms these features into vector embeddings that are suitable for machine learning model processing. Modern vision language models often use a Vision Transformer (ViT) [111], which adapts key elements of transformer-based language models to visual data. In this approach, an input image is split into a group of fixed-size patches. Each patch is embedded and treated as a token in a sequence. The model then applies self-attention to these patch tokens. This process helps the model understand the relationships between different areas of the image and create a transformer-based representation of the visual input.

2. **Projector:** The vision encoder produces a series of high-dimensional visual embeddings that are not yet directly compatible with the language model. The projector translates these embeddings into the embedding space of the language model, turning them into image tokens. This process allows the model to handle both visual and textual input together. The projector can be a simple linear transformation as in LLaVa [112] or a more complex cross-attention mechanism as in LLaMA-3.2-Vision [113].

Language Module: The language backbone is usually an LLM that processes textual tokens and generates the final output. Its architecture can follow any of the designs discussed in Section 2.3.2. In a VLM, the language model accepts both text tokens and projected image tokens, allowing for joint multimodal processing. Any standard LLM can be used for this purpose.

Fusion Mechanism: The fusion mechanism determines how visual and textual representations are integrated within a vision language model. Fusion strategies can be broadly divided into three main categories [114]:

1. **Early Fusion:** Visual and textual tokens are combined at the input stage and processed together by a single transformer backbone. This method merges different types of data at the feature level, meaning that raw visual and textual representations are combined before being sent to the model, allowing the model to capture detailed cross-modal relationships early in the process.

2. **Late Fusion:** Each modality is processed independently, with its embeddings combined only at the output stage. This allows each model to be optimized specifically for its modality.

3. **Mid-Level Fusion:** Cross-attention layers allow tokens from one modality to selectively attend to tokens from the other. This allows information to be exchanged dynamically during intermediate stages of processing. This method supports context-aware grounding, where the representation of each modality gains relevant features from the other. By combining the two streams at multiple points, cross-attention maintains a flexible balance between independent processing and complete joint modeling.

Table 7: Comparison of fusion mechanisms in VLMs.

Fusion Type	Advantages	Limitations	Example
Early Fusion	Captures fine-grained cross-modal interactions early; enables strong joint reasoning	Requires large-scale multimodal pretraining; higher computational cost	LLaVA [112]
Late Fusion	Efficient; allows independent optimization of each modality	Limited cross-modal reasoning; weaker generative performance	CLIP [115]
Mid-Level Fusion	Balances independent and joint processing; supports context-aware grounding	More complex architecture; may require careful tuning	Flamingo [116]

Table 7 lists the advantages, limitations, and examples for each fusion strategy.

2.4.2 Pretraining Strategies for VLMs

Training strategies for VLMs focus on aligning and integrating information from both the vision and language encoders. This allows the model to link visual content with textual descriptions and reason across the two modalities in a unified manner. Pretraining serves as the foundational stage, during which the model learns joint visual–textual representations before being adapted to specific tasks. The primary objective is to map visual features and linguistic concepts into a shared embedding space, enabling seamless cross-modal understanding. To achieve this, various pretraining goals have been developed. Each one targets different aspects of multimodal alignment and representation learning. Common approaches are:

1. **Contrastive learning:** The main idea of contrastive learning in VLMs is to bring similar image and text embeddings closer in the shared space while pushing dissimilar ones farther apart [117]. This is done using a similarity metric, with the model trained to minimize distances for matching pairs and maximize them for non-matching pairs.

2. **Masked image modeling:** Masked image modeling (MIM) involves obscuring parts of an image, such as pixels, patches, or latent representations. The model, often an autoencoder, is then trained to reconstruct the missing content using the visible context [118]. This idea gained attention in NLP with BERT [82], which showed the effectiveness of masking for learning contextual representations from large unlabeled datasets. In the visual domain, MIM uses the same principle by dividing an image into patches, masking a subset, and training the network to infer missing patches from the unmasked regions.

3. **Image-text matching:** Image-text matching (ITM) involves aligning visual and textual data in a shared embedding space to measure their compatibility. Images and text are first encoded into separate feature representations and projected into

a shared embedding space [119]. The model is trained to assign high similarity scores to matching pairs and low scores to non-matching pairs, often using cosine similarity. ITM is commonly used with contrastive learning to strengthen fine-grained cross-modal alignment.

4. **PrefixLM:** PrefixLM is a pretraining technique where the model receives a segment of text and predicts the next word in the sequence, and in VLMs, this extends to generating content conditioned on both an image and its corresponding text prefix [120]. A vision transformer turns the image into a sequence of patches, each patch representing a local area. These patches are then processed using convolution or linear projection to create contextualized visual embeddings. The text prefix is converted into token embeddings, and both visual and textual embeddings are fed into the transformer's encoder and decoder blocks, where the model learns the relationships between them to predict the rest of the text sequence.

5. **Multimodal fusing with cross-attention:** Cross-attention allows representations from one modality to connect with and merge information from another. In VLMs, visual features, such as image patch embeddings, and textual features, such as token embeddings, combine in cross-attention layers [121]. This allows the model to share context as it processes information.

6. **Knowledge distillation:** Knowledge distillation involves training a "student" model to mimic the behavior of a stronger "teacher" model [122]. The teacher produces "soft" output probability distributions for specific visual and text inputs. These outputs offer more detailed information than hard labels. The student learns to match these distributions, improving their understanding of both visual and textual features. This approach reduces the size of the model and the computation while keeping much of the teacher's performance.

2.4.3 Notable VLMs

Modern VLM architectures can perform complex multimodal reasoning across both visual and textual inputs. This makes them strong candidates for understanding diagrams. The following paragraphs discuss the models used in the data curation process and those assessed during the experimental phase for this thesis. Although all three models follow the general multimodal transformer architecture, they differ substantially, including in their vision encoders, fusion mechanisms, and training strategies. Table 8 summarizes the key differences.

Pixtral 12B: Pixtral 12B is a multimodal/vision language model trained to understand both natural images and documents [123]. It was released in 2024, as an extension of the Mistral architecture [124]. Pixtral follows the general VLM design described in Section 2.4.1, and includes a vision module, a language module, and a fusion mechanism. The high-level architecture of Pixtral 12B is shown in Figure 10.

Table 8: Comparison of Pixtral 12B, Gemma 3, and Qwen2.5-VL models.

Aspect	**Pixtral 12B**	**Gemma 3**	**Qwen2.5-VL**
Vision Encoder	400M PixtralViT trained from scratch; supports arbitrary resolutions	400M SigLIP encoder; fixed 896×896 input; outputs 256 visual tokens	Custom ViT with 2D RoPE; dynamic resolution; MLP patch merger
Fusion Strategy	Mid-level fusion (visual tokens injected at specific transformer layers)	Early fusion (visual and text tokens merged at input)	Merged/late fusion using multimodal RoPE
Positional Encoding	1D positional encoding inherited from Mistral backbone	1D positional scheme with mixed local/global attention	2D RoPE providing explicit spatial height–width grounding
Handling High Resolution	No tiling required; flexible aspect ratio support	Pan & Scan tiling for high-res or non-square diagrams	Native dynamic resolution with aspect-ratio preservation
Language Backbone	Mistral Nemo 12B; 128k context window	Gemma 3 decoder-only transformer; 128k context window	Qwen2.5 LLM decoder; strong multimodal grounding
Training Strategy	Image–text pretraining + instruction tuning	Large-scale distillation + multimodal pretraining + instruction tuning	Three-phase multimodal pretraining + post-training alignment

The vision module is a 400M parameter vision transformer (PixtralViT) that has been trained from scratch. It supports different image resolutions and aspect ratios without the need for fixed-size cropping or tiling. The projector is a fully connected two-layer network with GeLU activation [125] that maps the vision encoder output to the embedding space of the language model. The language module is based on Mistral Nemo 12B [126], which is a decoder-only transformer that has a 128k-token context window. The projected visual tokens are mapped into the same embedding space and positional encoding as text tokens. This setup allows the multimodal decoder to handle

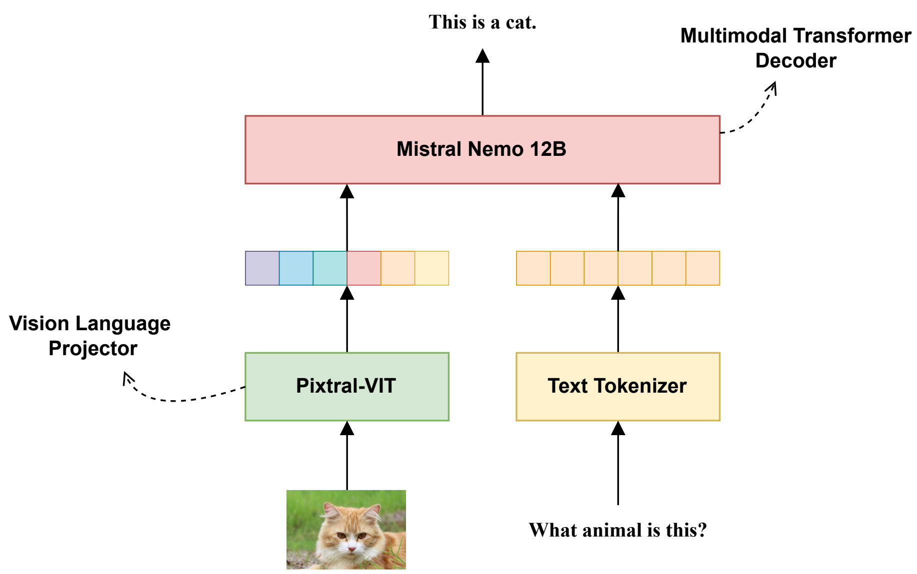


Figure 10: Architecture of Pixtral 12B.

them together in a single sequence. Pixtral adopts a mid-level fusion strategy, where visual tokens are introduced into the text stream at specific stages of processing. This method helps to achieve effective cross-modal grounding while keeping the visual feature extraction separate. The self-attention mechanism supports functions like multi-image conversations and combined reasoning during decoding.

Gemma 3: Gemma 3 is a family of open-weight large language models from Google DeepMind [127], ranging from 1B to 27B parameters. It was released in 2024 and was co-designed with the Gemini frontier models [128]. These models have several key capabilities, including multimodality, long-context processing, and multilingual support.

Gemma 3 uses a decoder-only transformer design similar in structure to Pixtral 12B, integrating a vision module and language module into a unified backbone. The general architecture of Gemma 3 models is shown in Figure 11.

The vision module is a 400M parameter version of the SigLIP encoder [129], a vision transformer trained with a variation of CLIP loss [130]. It processes images at a fixed resolution of 896×896 pixels and reduces the output to a fixed sequence of 256 soft visual tokens. This token representation functions as the output of a projector, mapping the embeddings of the vision encoder to a format compatible with the language model. The same vision encoder is used for the 4B, 12B, and 27B models. To manage non-square aspect ratios and high-resolution diagrams, Gemma 3 uses a Pan & Scan (P&S) method [127] during inference. This adaptive windowing algorithm

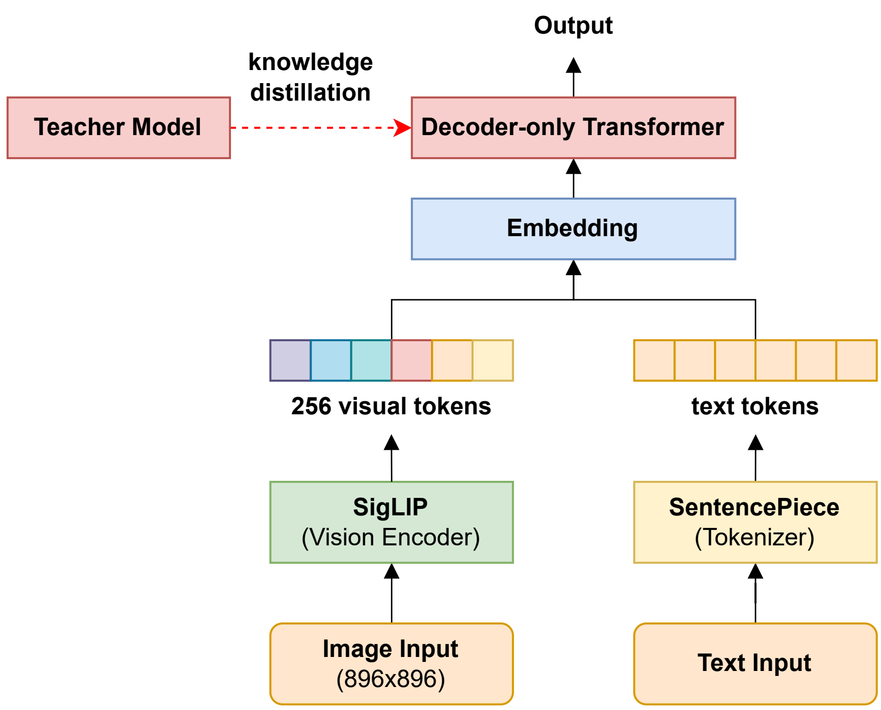


Figure 11: Architecture of Gemma 3 models.

divides the image into non-overlapping sections. Each section is resized to 896×896 before encoding. This ensures that fine details, such as small text and elements in diagrams, remain intact. The ablation results of the Gemma 3 technical report show that higher resolution encoders outperform smaller ones, and enabling P&S leads to significant improvements, especially for tasks that involve reading text from images. The language backbone of Gemma 3 is a decoder-only transformer designed for efficiency and long-context reasoning. It uses a mix of local sliding-window attention and global attention in a 5:1 pattern. Local layers focus on smaller spans of tokens, while global layers attend to the full context. This setup allows Gemma 3 to handle sequences of up to 128k tokens while keeping memory use manageable. The model also uses grouped-query attention [131] and root mean square layer normalization [132] to improve stability and performance. For image and text input, the 256 visual tokens produced by the SigLIP encoder are combined with text tokens at the input stage. This gives Gemma 3 an early fusion design that allows it to process visual and textual information together from the very first layer.

Gemma 3 models are trained using knowledge distillation from a larger teacher model. Pretraining uses a mix of text-only and image-text data. The token budgets are about 14T tokens for the 27B model, 4T tokens for the 4B model, and smaller budgets for the other variants. Images pass through a frozen SigLIP encoder, and visual embeddings are precomputed to avoid extra training time. The models use a 262k vocabulary SentencePiece tokenizer [133]. Extensive data filtering eliminates unsafe, low-quality, or duplicate content. Training takes place on large-scale TPU infrastructure [134], including TPUv4, TPUv5e, and TPUv5p, and uses high levels of parallelization. Following pretraining, the models go through instruction tuning. This is a focused fine-tuning process where the model is improved using high-quality instruction-response pairs [8]. Instruction tuning seeks to make the model's outputs match human expectations by teaching it to follow natural language instructions clearly and consistently. For Gemma 3, this process is designed to enhance reasoning, mathematics, coding, instruction-following, and multilingual understanding.

Qwen2.5-VL: Qwen2.5-VL is a vision language model developed by the Alibaba Cloud Qwen team [135]. It is designed to handle both image and video inputs within a unified multimodal framework. The high- level architecture of the Qwen 2.5-VL model is shown in Figure 12.

The vision encoder is a redesigned vision transformer trained from scratch. The vision encoder uses 2D Rotary Positional Embedding (RoPE) [136]. Unlike standard 1D RoPE, which text models use to encode the order of tokens in a line, 2D RoPE provides separate embeddings for height and width. This setup lets the model maintain the spatial layout of image patches. For instance, it can tell the difference between a patch in the top-left corner of a diagram and one in the bottom-right. Since images can create long sequences of patch features, Qwen2.5-VL introduces an efficient Multi-Layer Perceptron (MLP)-based merger before feeding visual tokens into the language model. Groups of four neighboring patches are combined and projected through a two-layer MLP into the same embedding space as text tokens. This design

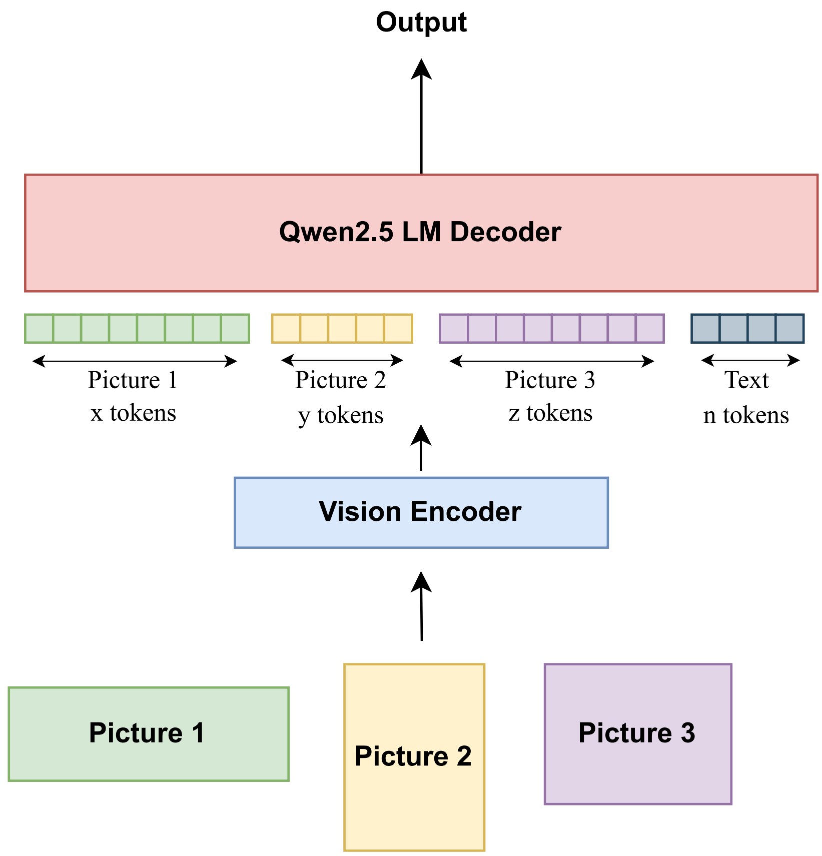


Figure 12: Architecture of Qwen 2.5-VL model

reduces computational cost while retaining essential visual details. The language backbone is initialized from the pretrained Qwen2.5 large language model, which is a decoder-only transformer. To support multimodal reasoning, it replaces the standard 1D positional embedding with a multimodal RoPE aligned to absolute time. This change allows for joint spatial and temporal grounding in text, images, and videos. Qwen2.5-VL also supports native dynamic resolution for images, which means that it can process them at their original scale without distorting their aspect ratios.

Qwen2.5-VL is trained in several stages to improve its multimodal capabilities. The pretraining follows a three-phase training recipe. In the first phase, only the vision encoder is optimized to match its features with the language model, using image-caption, OCR, and visual-knowledge data. In the second phase, all parameters are unfrozen, and the model is trained on a wider mix of multimodal datasets. The third phase increases the length of the sequence and adds more video and agent-based data, which helps the model to reason about longer contexts and temporal inputs. After pretraining, Qwen2.5-VL goes through post-training alignment. This stage sharpens the model's ability to follow instructions and ensures that the training corpus remains high quality.

3 Data

This chapter presents the dataset used in this thesis and the methodology and steps followed to construct it. The creation and development of a high-quality dataset was crucial for conducting meaningful experiments and obtaining reliable results. Section 3.1 provides an overview of the dataset creation process. Section 3.2 offers a detailed explanation of the image curation process, while Sections 3.3 and 3.4 describe the design and generation of textual context and questions for Visual Question Answering (VQA), respectively. Finally, Section 3.5 provides a summary of the final dataset.

3.1 Overview of Methodology for Dataset Creation

The objective of this thesis is to assess whether state-of-the-art VLMs can effectively interpret UML sequence diagrams, both with and without supplementary textual context. To achieve this, a custom private dataset was constructed using private technical documents to retrieve and compile useful images, and to generate context as well as questions related to these images to evaluate the models.

Due to the confidentiality of the source material, raw diagrams, code, and document excerpts cannot be disclosed. However, for clarity and ease of understanding, Figure 13 provides an anonymized sample representation of the types of UML sequence diagrams in this thesis. Other sample images are given in Appendix A.1.

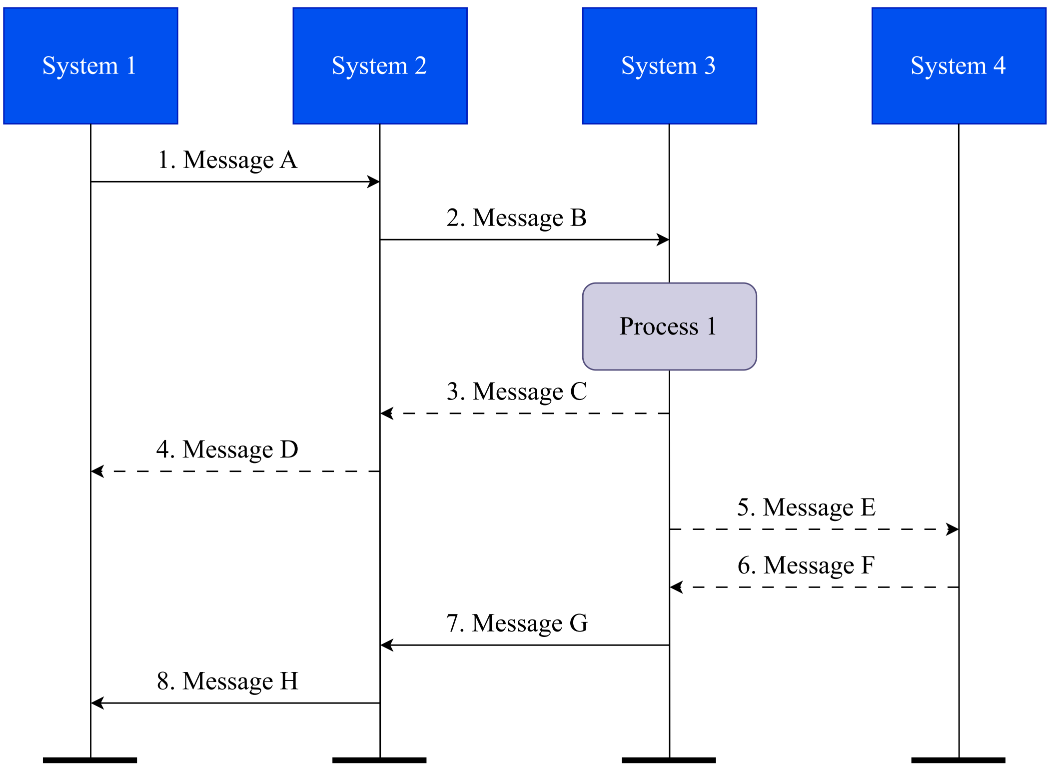


Figure 13: Anonymized example of a UML sequence diagram used in this thesis.

As discussed in Section 2.1.3, UML sequence diagrams depict interactions between system components over time. The diagrams span a range of complexity, and the one shown reflects a mid-level example. While the original content remains private, the full dataset construction methodology is documented in detail in the following sections.

It is important to highlight that the experiments conducted in this thesis cannot be replicated with public data as there is no publicly available dataset at the time of writing that matches the structure of the training and evaluation criteria for this thesis. Using private datasets also ensures that the risk of model data leakage remains low. This is important as even limited exposure to such data during training can lead to inflated evaluation scores and misleading conclusions about model performance [137].

The dataset creation process followed a three-stage pipeline to generate its components:

1. **Image Extraction and Curation:** Relevant UML sequence diagrams were systematically retrieved from documents.

2. **Context Generation:** For each diagram, textual context was generated to study its effect on model performance during querying.

3. **Question and Answer Generation:** Three multiple-choice questions were created per diagram to probe model capabilities across different tasks, each with clearly defined answer options.

Figure 14 provides an overview of the dataset construction process. The final dataset is composed of three interdependent components (image, context, and question/answer pair). The context was generated based on the image, while the questions were formulated using both the context and the image. Each component was manually vetted to ensure high-quality data. The steps are explained in detail in the following sections.

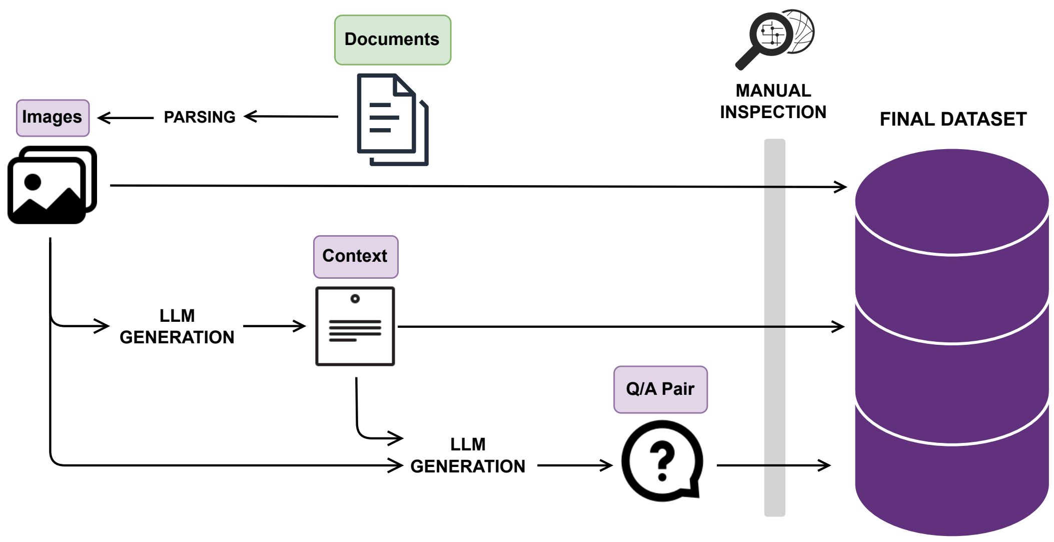


Figure 14: Overview of the dataset components and construction process

3.2 Image Dataset Preparation

In this section, a detailed review of the steps involved in preparing and curating images from the data set is given. An overview of the full image dataset preparation pipeline is illustrated in Figure 15. The images were initially parsed from the documents as described in Section 3.2.1. The images were then classified according to their type and complexity, given in Sections 3.2.2 and 3.2.3, respectively. In the end, duplicates were removed and the final dataset was manually vetted to ensure quality and consistency, as discussed in Section 3.2.4.

3.2.1 PDF Parsing and Preprocessing

The private internal data, initially in PDF format, required extensive preprocessing. A total of 775 PDFs with 130,670 pages across all PDFs of private data were parsed to extract embedded images. Initial analysis of the PDF contents revealed four distinct types of colourspaces used in the embedded images:

1. DeviceRGB: A standard RGB colourspace representing red, green, and blue channels.
2. ICC Based: A colorspace based on embedded International Color Consortium (ICC) profiles.
3. DeviceGray: A grayscale colorspace, typically used for black-and-white images.
4. Indexed: A colorspace where colors are stored in a lookup table and are referenced via index values instead of full definitions.

All colourspaces are part of the PDF specification defined by Adobe Systems [138].

Initial Image Extraction and Filtering: The PDF parsing process began with the help of a Python script utilizing libraries for image extraction. To filter out small, irrelevant images (e.g., icons, logos, blank images), a size threshold of 3 kB was used. Images smaller than this threshold were discarded. This filtering step also eliminated all DeviceGray images, as they were found to be either blank pages or transparent layers of other image types, thus not relevant to the study. After this initial filtering, a total of 9,493 images were extracted from the PDFs, comprising of DeviceRGB, ICCBased, and Indexed colorspace types.

Removal of Non-Semantic Gradient Backgrounds: To further refine the dataset, an additional filtering technique was implemented, focusing primarily on the DeviceRGB images. Among the DeviceRGB images, several were identified as background templates, often used as title slides or section headers, and were characterized by colored gradients and no meaningful content. These images were mainly of four specific resolutions (1994×798, 4000×1240, 1820×878, 1656×798). A pixel-color matching technique was used to identify and exclude images with similar edge

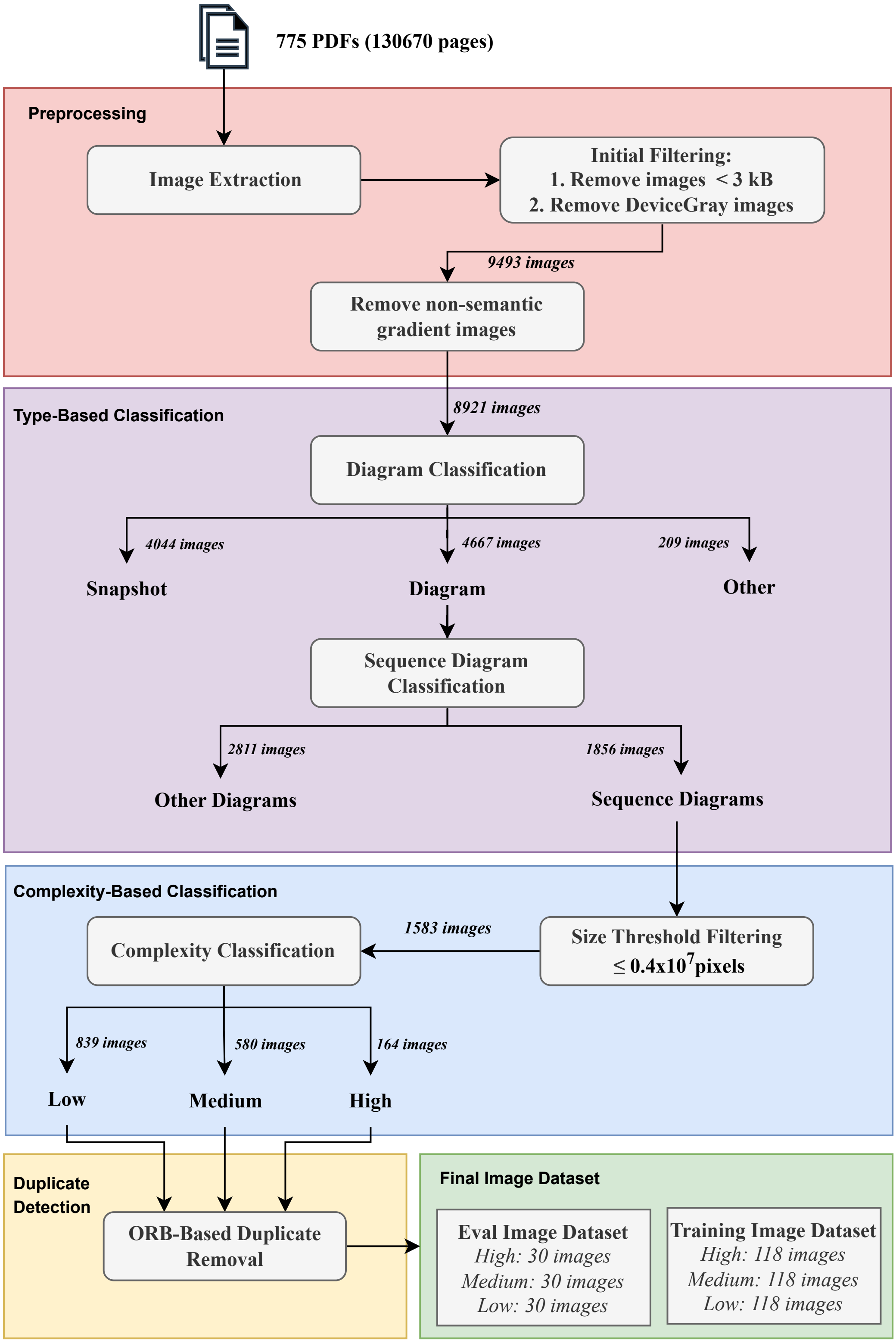


Figure 15: Overview of the image dataset preparation workflow.

colors (sampled at a 50-pixel offset from the image borders). This step eliminated an additional 572 images, leaving 8,921 images for further processing.

3.2.2 Type-Based Classification of Images

After the initial filtering steps, the remaining dataset comprised a diverse range of image types. These included user interface screenshots, various diagrams, code snippets, mathematical formula renderings, and a small number of decorative or empty visuals that were not removed earlier. Within the diagrammatic content, the most common categories included message service request diagrams, architectural network diagrams, and occasional flowcharts. The objective of the classification process was twofold:

1. Distinguish diagrams from all other image types, such as screenshots and decorative elements.

2. Identify and isolate UML sequence diagrams from the subset of images already classified as diagrams.

Each stage employed a different prompting strategy, as outlined in the sections below.

Diagram Classification: To effectively separate diagrams from other image types, a manually annotated test set of 90 randomly sampled images was created. These images were evenly distributed across three categories: `diagram`, `snapshot`, and `other`, with 30 images in each class.

For automated classification, the Pixtral 12B multimodal vision-language model [123] was employed due to its:

1. Open-source nature with a permissive Apache 2.0 license [139], making it suitable for internal and commercial use by the company.

2. Instruction-following capabilities.

3. Support for variable image sizes and aspect ratios.

4. Compatibility with local inference through the vLLM library, an efficient framework for LLM inference and serving developed by the Sky Computing Lab at UC Berkeley [140].

The annotated test set was used to refine and validate the classification prompt. Through iterative testing, the prompt was optimized until the model achieved consistent performance. In two separate evaluation runs, Pixtral 12B correctly classified all 90 images, achieving 100 % accuracy. Based on this result, one could be confident in the model's ability to generalize across the broader set of images. Given the relatively straightforward nature of the classification task and the strong performance of Pixtral 12B, the use of a larger model was deemed unnecessary, as it would only introduce additional computational overhead.

Applying the classifier to the full dataset yielded the following distribution:

Classify the image into one of the following categories:

1. Diagram – A structured visual representation of a system, process, or workflow. This includes flowcharts, architectural design diagrams, high-level design diagrams, and sequence diagrams that illustrate components, relationships, interactions, and responsibilities across different entities or systems.

2. Snapshot – A captured image of a software interface, document, or digital content, typically showing UI elements such as buttons, menus, or dropdowns. This includes images showing application interfaces, webpages, dashboards, or any other digital platform layout.

3. Other – Any image that does not fit into the above categories, such as photos, illustrations, or general graphics.

Carefully examine the visual elements, structure, and purpose of the image to determine the most appropriate classification and give your answer in only one word: diagram, snapshot, or other. There should be no other text.

Figure 16: Prompt for classifying images into either diagrams, snapshots, or other.

1. 4,667 images classified as diagrams,
2. 4,044 as snapshots or screenshots,
3. 209 as other.

Only a single image was not confidently assigned to any of the three predefined categories, resulting in an effective classification success rate of approximately 100 %.

The optimized prompt used for diagram classification is shown in Figure 16.

UML Sequence Diagram Classification: Following the isolation of diagrams from the broader dataset, the second classification task focused on identifying UML sequence diagrams within the subset of images already labeled as diagrams. The objective was to distinguish these from other diagram types, including architectural diagrams, hardware schematics, flowcharts, and various other types used in documentation. To assess the performance of Pixtral 12B for this task, a test set of 100 randomly sampled diagram images was created. Of these, 41 were labeled as UML sequence diagrams and 59 were labeled as non-UML sequence diagrams. Despite multiple prompt refinement attempts and accuracy exceeding 90 %, the model's performance was deemed unsatisfactory due to significant misclassifications of certain images. These misclassification posed potential risks for downstream training and evaluation tasks.

To address this limitation, a few-shot prompting approach was adopted, providing the model with one labeled UML sequence diagram and two labeled non-UML examples as context. Additionally, a more robust model, Gemma3 27B-IT [127], was leveraged, which offered excellent support for variable image sizes and aspect ratios, and strong few-shot learning capabilities.

You are given an image that is already classified as a diagram. Now, further classify it into one of the following two categories:

1. UML_SR – This refers specifically to UML sequence diagrams that illustrate service request interactions in telecom or software systems. These diagrams follow the UML standard for representing time-ordered exchanges between system components or network functions. They typically include:
- Vertical lifelines representing system modules (e.g., DB, UDM)
- Horizontal arrows labeled with messages or function calls (e.g., 'Namf_EventExposure_Subscribe', 'Notify', 'Update Location Request')
- Optional annotations like '201 Created', '200 OK', or database actions like 'DB Read/Update'
These diagrams are used to show the step-by-step flow of operations across distributed systems or services.

2. Other – Any diagram that is not a UML sequence diagram representing service request flows. This includes:
- Architecture diagrams showing system layouts or high-level components
- Flowcharts illustrating processes or decision paths
- Data models, entity-relationship diagrams, or any visual representation that does not depict ordered inter-component messaging over time

Carefully examine the image for the presence of lifelines, ordered message flows, time progression, and labeled interactions characteristic of UML sequence diagrams. Pay special attention to whether the diagram represents a time-sequenced interaction between components. If the image instead shows hardware, ports, or static layout with labels, classify it as 'Other'.

Based on this analysis, classify the image strictly into one of the two categories.
Give your answer in only one word: UML_SR or Other. There should be no other text.

Figure 17: Instruction prompt for classifying diagrams as UML_SR or Other.

Prompt Used for UML Sequence Diagram Classification: The few-shot prompt used for image classification was constructed by combining a base instruction with a sequence of labeled image examples, followed by the query image to be classified. The base instruction in Figure 17 was provided in the initial user message and was used to guide the classification logic of the model.

The few-shot prompt format in Figure 18 shows the prompt structure that utilizes role-based dialogue and image-text pairs. Each "user" input consisted of an image and a prompt, while the "assistant" responded with a one-word classification label.

The few-shot setup, combined with the improved capabilities of Gemma3 27B-IT, significantly enhanced classification performance. As shown in Table **??**, the model correctly identified 99 out of 100 images, with only a single misclassification.

Applying this prompt to the 4667 diagram images yielded the following result:

1. 1,856 images identified as UML sequence diagrams (`UML_SR`).

2. 2,811 images classified as other diagrams.

System: You are a helpful assistant.

User: `<base_instruction>`
`<image1>`
"Classify this diagram."
Assistant: "UML_SR"

User: `<image2>`
"Classify this diagram."
Assistant: "Other"

User: `<image3>`
"Classify this diagram."
Assistant: "Other"

User: `<image4>`
"Classify this diagram."

Figure 18: Few-shot prompt structure for sequence diagram classification.

Table 9: Comparison of Pixtral 12B (zero-shot) and Gemma3 27B-IT (few-shot) on UML sequence diagram classification.

Metric	Pixtral 12B	Gemma3 27B-IT
True Positives (UML as UML)	37	40
False Positives (Other as UML)	2	0
False Negatives (UML as Other)	4	1
True Negatives (Other as Other)	57	59
Accuracy	0.940	0.990
Precision	0.949	1.000
Recall	0.902	0.976
F1-score	0.925	0.988

3.2.3 Complexity-Based Classification of Images

Following the successful identification of UML sequence diagrams, the next phase involved categorizing these images based on their visual and structural complexity. Each image was assigned a label of *low*, *medium*, or *high* complexity, according to self-defined criteria.

This classification served as a final filtering step before model training and downstream evaluation. The primary motivations for this classification were the following:

1. **Dataset Diversification for Fine-Tuning:** To prevent bias during model training, it was essential to construct a training dataset that includes a balanced mix of diagrams across different complexity levels. This ensures that the model does not overfit to simpler diagrams or underperform on more intricate ones.

2. **Complexity-Aware Evaluation:** By curating a representative evaluation set spanning all three complexity levels, we aimed to assess whether model performance varies with diagram difficulty. This enables us to assess whether the inclusion of contextual information leads to improved accuracy on higher-complexity diagrams.

Criteria for Complexity: To systematically categorize UML sequence diagrams by complexity, a clear and consistent set of rules was developed. The criteria needed to generalize well across the dataset while remaining specifically tailored to the structure and characteristics of UML diagrams. The following rule-based framework was adopted for reliable classification and the complexity levels were defined as follows:

1. **Low Complexity:** Diagrams with 2 to 4 lifelines or fewer than 8 messages.
2. **Medium Complexity:** Diagrams with 5 to 6 lifelines or between 8 to 15 messages.
3. **High Complexity:** Diagrams with 7 or more lifelines or more than 15 messages.

If the number of lifelines and messages fall into different categories, the diagram is assigned to the higher complexity class to ensure conservative labeling. Figure 19 presents schematic outlines of sequence diagrams of the different complexities.

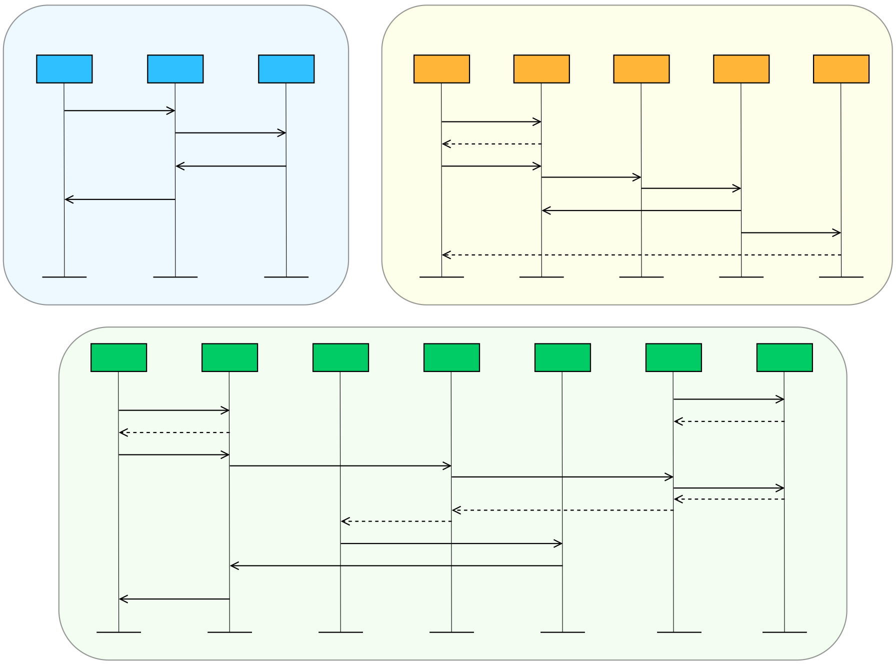


Figure 19: Schematic outlines of sequence diagrams of each complexity. The diagrams are colour-coded as follows: blue for low complexity, yellow for medium complexity, and green for high complexity.

```
You are given a UML sequence diagram that illustrates interactions between system components over time. Your task is to classify the overall complexity of the diagram as low, medium or high. Follow these steps before deciding:

<base_instruction_and_criteria>

At the end, return only one word as your final answer: low, medium or high.

<few-shot examples (3)>
<test image>

Classify this diagram.
```

Figure 20: Few-shot prompt structure for complexity classification.

Sample Study for Complexity Classification: Once the complexity criteria were established, the next step was to evaluate whether a VLM could reliably apply these rules across the filtered set of 1,856 UML sequence diagrams. To do so, a randomly selected sample of 100 images was manually annotated according to the defined complexity levels: *low*, *medium*, or *high*. This annotated subset served as a benchmark for validating the generalizability of the rule-based framework. The final label distribution of this sample consisted of 52 low-complexity diagrams, 36 medium-complexity diagrams, and 13 high-complexity diagrams.

Few-Shot Prompting Strategy: To assess whether multimodals could generalize the complexity classification task, two models were tested: Gemma3 27B-IT and Qwen2.5-VL 72B [135]. Both models were tested using a few-shot prompting setup, where each prompt contained three labeled examples, one from each complexity level: low, medium, and high, followed by the image to be classified. Each of the 100 annotated images was processed individually using this approach. The prompt began with a clearly defined instruction block outlining the complexity criteria, followed by the three reference examples to guide the model's response. A generalized version of this prompt structure is shown in Figure 20. The complete base instruction is available in Appendix B.1

Performance on Sample Set: The performance of the two models was evaluated on the manually annotated 100-image sample set. For Gemma3 27B-IT, the model achieved an overall accuracy of 75 %, while Qwen2.5-VL 72B yielded an accuracy of 69 %. The confusion matrices in Table 10 illustrate the classification breakdown for each model.

While neither model achieved perfect accuracy, a key observation is that no high-complexity diagrams were misclassified as low, and vice versa. Most misclassifications occurred between adjacent categories, specifically, between low and medium or medium and high. This suggests that the model's decisions were often near the decision boundary. Upon inspection, many of the borderline cases exhibited overlapping visual traits that aligned with multiple complexity levels, contributing to classification ambiguity.

Table 10: Results for sample complexity classification.

Gemma3 27B-IT			
Actual \ Predicted	**Low**	**Medium**	**High**
Low	37	14	0
Medium	2	31	3
High	0	6	7
Qwen2.5-VL 72B			
Actual \ Predicted	**Low**	**Medium**	**High**
Low	34	17	0
Medium	6	27	3
High	0	5	8

Although the two models produced comparable classification results, Gemma3 27B-IT emerged as the preferred option due to its significantly lower inference time, completing the classification in approximately 110 seconds for all 100 samples, compared to over 870 seconds required by Qwen2.5-VL 72B. Given this performance-to-efficiency trade-off, Gemma3 27B-IT was selected as the final model for performing inference on the complete dataset.

Dataset Refinement for Size Uniformity: To address the potential confounding effects of image size on complexity classification, a preliminary analysis was performed on the same subset of 100 randomly selected samples referenced earlier. This analysis aimed to determine whether image resolution, defined as the total pixel count (width $\times$ height), correlated significantly with the assigned complexity labels.

The analysis indicated a minor increase in average image size with rising complexity levels; however, the difference was insufficient to suggest a strong or consistent relationship. As shown in Figure 21, the box plots reveal that several low-complexity and medium-complexity images exhibit higher pixel counts than many high-complexity samples. Notably, these outliers extend beyond the upper quartile range of the high-complexity group, further suggesting that image size is not a reliable indicator of complexity.

Thus, we can remove outliers from our training and evaluation data to maintain consistency. Based on this observation, a standardized image size threshold of $\leq 4 \cdot 10^6$ pixels was selected to ensure uniformity throughout the dataset. Within the subset of 100 images, 90 samples satisfied this criterion. Applying the same threshold to the full dataset reduced the number of usable images from 1,856 to 1,583.

Inference on Full Dataset: The filtered set of 1583 images was then subjected to final complexity classification using the selected Gemma3 27B-IT model and the previously defined few-shot prompting strategy. As a result, the dataset was stratified

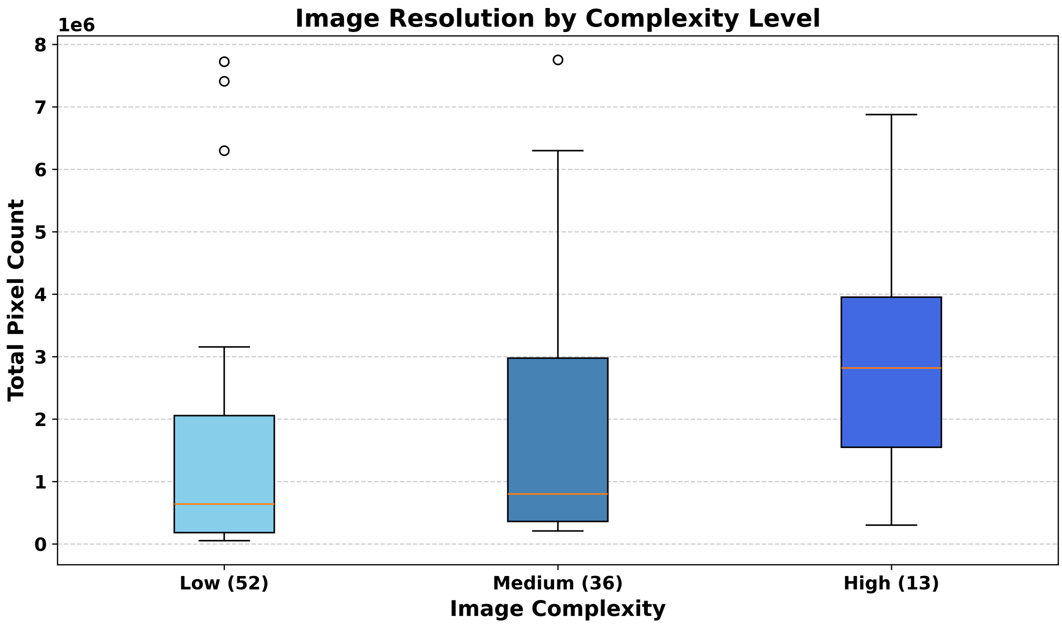


Figure 21: Box plots showing the distribution of image sizes (in pixel count) across different complexity levels.

into three complexity categories: 839 images labeled as *low*, 580 as *medium*, and 164 as *high* complexity.

3.2.4 Duplicate Detection and Clustering

Many diagrams extracted from the documents were repeated in different versions with very slight alterations. These near-duplicate diagrams posed a risk of data leakage, as the model could be fine-tuned on one version and evaluated on another, leading to artificially inflated performance. To mitigate this, an image similarity filtering mechanism was applied based on ORB, a feature detection and matching technique, whose workflow is illustrated in Figure 6, to detect and remove duplicates from the dataset.

To implement the ORB-based duplicate detection pipeline, two threshold values were used. The first threshold, set to 20, defines what counts as a "good" match between two keypoints in the image: if the distance between two ORB descriptors is less than 20 (based on the Hamming distance [141]), the match is considered valid. The second threshold is set to 40. This determines if two images are near-duplicates. If more than 40 good matches are found between a pair of images, the pair is flagged as a duplicate. These thresholds were chosen after manually inspecting several pairs of images. This ensured a balance between accurately identifying duplicates and avoiding false matches. The complete detection and clustering procedure is described in Algorithm 2.

Algorithm 2 Duplicate Detection Using ORB Features

Require: Set of input images $\{I_1, I_2, \ldots, I_n\}$; thresholds t_{match}, t_{dup}

for all images I_i **do**
    Convert to grayscale
    Extract ORB keypoints and descriptors
end for
Generate all unique image pairs (I_i, I_j)
for all pairs (I_i, I_j) **do**
    Match descriptors using Hamming distance
    Identify good matches with distance $< t_{\text{match}}$
    if number of good matches $> t_{\text{dup}}$ **then**
        Mark (I_i, I_j) as near-duplicate
    end if
end for

3.3 Textual Context Generation

To evaluate and study the impact of textual context on VLMS, structured context descriptions were generated for each image in the dataset. The context served 2 primary purposes:

1. To provide supplementary information for VLMs to use during inference.
2. To support the formulation of questions in the subsequent step, given in the next section.

For each diagram, two types of textual context were produced, detailed context and sparse context. Their definitions are given in Table 11.

3.3.1 Generation Process and Prompts

By generating both detailed and sparse context, the objective was to assess whether the performance gains achieved with the detailed context justified the additional context

Table 11: Definitions of sparse and detailed context used for VLM evaluation.

Context Type	Definition
Sparse Context	Mentions components only as labeled (no abbreviation expansion). Does not describe message flows, sequencing, or control logic. Does not explain the function of any system element. Typical length: 1–2 sentences.
Detailed Context	**Sprase Context** + Mentions all lifelines or participants, including full forms of abbreviations. Describes two or more message flows (or all, if only 1–2 exist). Refers to at least one visible logic structure such as a condition or loop. Minimum length: 4 sentences.

Prompt for Sparse Context Generation

`<image>`
Provide a description or caption for the diagram that describes the general process or system being illustrated. You may mention any components shown in the diagram, but only use them exactly as labeled. Do not expand or explain them. Do not describe any message flows, sequence of interactions, or control logic.
Do not include introductory phrases. Avoid conversational or markdown format. The abbreviations listed below may be useful for understanding the components in the diagram:
`<JSON object with list of abbreviations>`

Prompt for Detailed Context Generation

`<image>`
Write a detailed and structured explanation of the diagram. Organize your response using clear sections.
- Main Components: List all systems, lifelines, or components involved. Expand any abbreviations used in the diagram.
- Sequence Flows: Describe the sequence of 2 or more message exchanges, including senders and receivers. The sequence flows should be a numbered list with each bullet as #<number>.
- Conditional Logic: Mention at least one visible logic structure, such as conditions, loops, or branches.
- Other Observations (optional): Mention any other relevant elements shown in the diagram.
Do not include introductory phrases like "Here is". Avoid conversational or markdown format. The abbreviations listed below may be useful for understanding the components in the diagram:
`<JSON object with list of abbreviations>`

Figure 22: Prompts used to generate spares and detailed textual contexts from sequence diagrams.

or verbosity in the inputs. The generation of the context was performed using the Claude 3.5 Sonnet v2 model [142], hosted internally by the company. To improve grounding, especially for private terminology, the prompts were appended with a JSON object at the end that contained the full forms of all the abbreviations used in the documents. This allowed the model to interpret unfamiliar and/or complex terms better. Figure 22 shows the prompts for generating both sparse and detailed context. The outputs generated using the sparse prompt were combined with those produced using the detailed prompt for the detailed context. All generated contexts were manually reviewed to remove errors that could prove costly in evaluation, as well as for question generation in the subsequent step. Appendix A.2 provides outputs of contexts for sample images to help understand the quality and characteristics of the generated contexts.

3.4 VQA Question Design and Generation

To evaluate the proficiency of VLMs in understanding sequence diagrams and investigate the impact of contextual information on their performance, three types of questions were designed, inspired by the FlowCE benchmark [143]. These questions

Task Category	Sample Question Templates
Information Extraction	What is the text content in the [color/shape] box? What information is contained in the note attached to [component/system]? What is all of the extracted textual information from [specific area] of the diagram?
Location Recognition	What is the position of [component 1] in relation to [component 2]? What message or action immediately follows [Specific Node/Action]? What is the step or message that precedes [Specific Node/Action]?
Sequence Flow Analysis	Which arrow represents the [Message X] flow? What is the direction of the [Message Y] arrow? Which component sends [Message Z] to which other component?

Table 12: Example questions per task category used for model inference

were designed to assess basic perception tasks. All questions have four answer options with one correct answer. Performance was measured using the Exact Match (EM) metric. The three task categories are:

1. **Information Extraction (IE):** These questions evaluate the ability of a model to extract specific textual information from a diagram.
2. **Location Recognition (LR):** These questions assess a model's ability to comprehend the spatial and positional relationships among various entities and message flows within a system.
3. **Sequence Flow Analysis (SA):** These questions specifically test a model's ability to analyze message sequence diagrams, focusing on sequence flows, conditional logic, and arrow directions.

Table 12 shows templates of question examples for each category of tasks.

3.4.1 Generation Process and Prompts

All questions were generated using both the image and the detailed context (which the sparse context is a subset by definition) to ensure a coherent question formulation. For each image, three questions were initially generated for each of the three tasks mentioned above. For information extraction and location recognition, the model was explicitly instructed to formulate questions that would not directly quote the context, preventing potential data leakage. In the end, questions for the sequence flow analysis were generated. The model was guided to use one specific sequence flow or conditional logic from the context to generate one question. The flow or logic was noted and subsequently removed from the context afterwards to mitigate data leakage concerns during inference.

Base Instruction for Information Extraction

You are an AI assistant specialized in analyzing message sequence diagrams and generating questions about textual information within them. Given the following message sequence diagram and its context, please generate 3 multiple-choice questions with four options each (out of which one is correct) that test the ability to extract specific textual information from the diagram.

Base Instruction for Location Recognition

You are an AI assistant specialized in analyzing message sequence diagrams and generating questions about spatial relationships within them. Given the following message sequence diagram and its context, please generate 3 multiple-choice questions with four options each (out of which one is correct) that test understanding of the positional relationships between entities and message flows.

Base Instruction for Sequence Flow Analysis

You are an AI assistant specialized in analyzing message sequence diagrams, particularly focusing on sequence flow, conditional logic, and arrow directions. Given the following context for a message sequence diagram, please generate 3 multiple-choice questions with four options each (out of which one is correct) that test understanding of these aspects within the diagram.

Prompt Template for All 3 Tasks

```
<image>
<base instruction>
<detailed context>
<rules>
<example questions>
<output format template>
```

Figure 23: Base instructions and general prompt template used for question/answer generation

The prompts were crafted by combining a base instruction with rules, example questions, and an output template for each. The base instruction varied for each question, but the overall prompts were largely similar. Figure 23 shows the base instruction for each task and shows the overall prompt format used in the generation process. The generation of the questions and answesr was performed using the same Claude 3.5 Sonnet v2 model as the one used for generating the context in the previous step. A comprehensive view of all the prompts can be found in Appendix B.2.

Each prompt resulted in three questions per task. From these, a question was randomly selected for the final dataset to ensure diversity. All selected questions were manually inspected to ensure high-quality questions. In case of any issues, the problematic questions were replaced with suitable alternatives from the initial generation set. This ensured a diverse, unbiased, and high-quality question set for evaluating the VLM's performance for our tasks. Appendix A.3 presents examples of questions generated for sample images, providing a clearer understanding of the data.

Table 13: Training dataset overview.

Complexity	Images per Class	Questions per Image	Total Samples
Low	118	3	354
Medium	118	3	354
High	118	3	354
Overall	**354**	**3**	**1062**

Table 14: Evaluation dataset breakdown by task type and complexity.

Task Type	Complexity	QA Pairs
Information Extraction	High	30
	Medium	30
	Low	30
	All	**90**
Location Recognition	High	30
	Medium	30
	Low	30
	All	**90**
Sequence Flow Analysis	High	30
	Medium	30
	Low	30
	All	**90**
Overall	High	90
	Medium	90
	Low	90
	All	**270**

3.5 Final Dataset Summary

This section gives a summary of the final dataset. Each row in the dataset consists of an image, a sparse context, a detailed context, and one question with four options for each of the three types of tasks. Breakdowns of the training and evaluation dataset splits are presented in the following sections.

3.5.1 Training Set Composition

The training portion of the dataset supports fine-tuning of VLMs on the three tasks. Each complexity level (low, medium, and high) contains 118 images. This results in 354 unique images for the training set. For each image, three questions were provided, one corresponding to each particular task: information extraction, location recognition, and sequence flow analysis. Altogether, the training set contains 1062 question-answer pairs for the training set and sets up the foundation for supervised fine-tuning experiments. Table 13 outlines the distribution of images, the number of questions per image, and the total number of samples at each complexity level.

3.5.2 Evaluation Set Composition

The evaluation portion of the dataset serves as the basis for all inferences in this thesis. It is evenly divided into three complexities. For each image, three questions are provided, one for each type of task. Thus, each complexity group has 30 samples per task type, making 90 samples for each task type and 90 samples for each complexity level. This allows for the separation of performance analysis per task and complexity. Table 14 gives a detailed breakdown of the number of QA pairs by task type and complexity.

4 Methodology and Experiments

This chapter outlines the methodology and experimental design adopted in this thesis. Section 4.1 introduces the research approach and defines the experimental dimensions, linking them to the research questions. Section 4.2 presents the experimental design while Section 4.3 details the fine-tuning process, covering LoRA configurations and training parameters. Section 4.4 explains the construction of evaluation prompts for different textual contexts and few-shot settings. Section 4.5 describes the evaluation and inference setup used to assess the performance of the models under all conditions.

4.1 Research Approach

In this thesis, the goal is to investigate the influence of textual context, few-shot learning, and fine-tuning on the performance of vision language models (VLMs) in understanding UML diagrams. The aim is to test the models on basic perception tasks, to test the vision capability of VLMs, and to determine whether different prompt strategies (zero, few-shot) or training strategies (baseline, fine-tuned models) enhance model performance.

The dataset used for this research is given in detail in Chapter 3, along with examples provided in Appendix A. The study is structured around three types of core perception tasks with the exact definitions and example question templates for each task provided in Section 3.4. These tasks were specifically chosen to evaluate the model in situations that test its perception capabilities, where the language component alone is insufficient. Although the language part can aid in reasoning in these tasks, it cannot consistently answer questions without analyzing the visual content of the UML diagrams. To answer these questions, the experimental and evaluation framework for this thesis is structured around five key dimensions:

1. **Textual Context:** For each task, the VLMs are tested for three types of textual context: no context, sparse context, and detailed context. In each case, the context is provided with the image and question-answer pair of each task. The definitions and descriptions of the types of context are given in Section 3.3. This framework enables the assessment of the dependence of model performance on textual cues, thereby highlighting the influence of the language component of the model. These experiments address the research question RQ1.

2. **Prompt Format:** All tasks are evaluated for both zero-shot and few-shot prompt formats. Few-shot prompts include three-shot and nine-shot. This structure helps in understanding whether examples in prompts aid in improving the model's ability to understand and answer questions. This helps in finding the answer to the research question RQ2.

3. **Training Strategy:** The vision language models are fine-tuned independently for each type of context: no context, sparse context, and detailed context, resulting in three separate fine-tuned models. Each of these fine-tuned models along with the base model is evaluated under all combinations of textual context (no

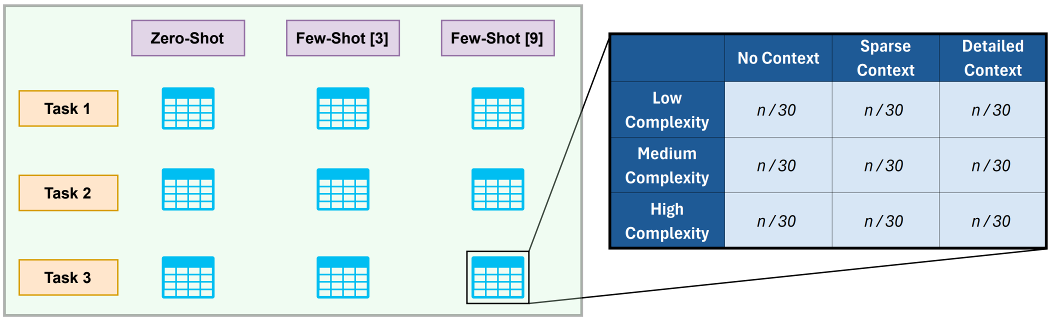


Figure 24: Overview of the experimental framework. Each row in the outer grid (on the left) represents one of the three task types, and each column corresponds to a prompt format (zero-shot, few-shot with 3 images, few-shot with 9 images). Each cell contains a 3×3 matrix representing combinations of textual context (none, sparse, detailed) and diagram complexity (low, medium, high). This structure is replicated for each model variant (baseline and fine-tuned).

context, sparse context, and detailed context) and prompt formats (zero-shot, three-shot, nine-shot) to analyze whether fine-tuning improves a model's ability to perform basic perception tasks, directly addressing research question RQ3.

4. **Task Type:** As mentioned previously, the evaluation focuses on three types of tasks. The structure of these tasks remains the same across all experimental settings. However, the results are analyzed separately for each task. This helps to see how different task types respond to various combinations of context, prompting, and fine-tuning, providing analysis for research question RQ4.

5. **Diagram Complexity:** Each diagram in the dataset is labeled as low, medium, or high complexity based on specific guidelines as outlined in Section 3.2.3. To support diversity during training, the few-shot examples and fine-tuning datasets include a balanced mix of diagrams from all three complexity levels. However, during the evaluation, the results can be analyzed separately by complexity group. This approach allows for a more focused investigation into the research question RQ5.

Figure 24 demonstrates how the experimental framework supports evaluation in all key dimensions: prompt format, textual context, diagram complexity, and task type. The template in the figure is identically applied to the baseline and fine-tuned models.

4.2 Experimental Design

The experiments in this thesis were designed to evaluate vision language models across the multiple dimensions outlined in the preceding section. Gemma 3, introduced in Section 2.4.3, was selected for these experiments due to its strong compatibility with frameworks used for fine-tuning and evaluation. The model supports both image and text input, making it suitable for the tasks in this thesis.

Table 15: Specification of Gemma 3 models used.

Feature	**Specification**
Vision encoder	SigLIP-based
Token context length	128k tokens
Visual tokens per image	256 tokens
Vocabulary size	256k tokens
Language model parameters	**4B**: ~4 billion | **27B**: ~27 billion

Gemma 3 offers several size options that share an identical multimodal architecture, with the only difference being the size of their language model component. This ensures that any performance variations are due to parameter scaling rather than changes in the vision encoder or other components. The key specifications for the baseline models evaluated in this thesis, Gemma 3 4B and Gemma 27B, are summarized in Table 15.

Building on this shared architecture, several model variants were evaluated to study the impact of fine-tuning. The following Gemma models were evaluated:

1. Gemma 3 4B.
2. Gemma 3 27B.
3. Gemma 3 4B fine-tuned; trained on images without textual context.
4. Gemma 3 4B fine-tuned; trained on images with sparse textual context.
5. Gemma 3 4B fine-tuned; trained on images with detailed textual context.

Each model, both baseline and fine-tuned, was tested under the same experimental conditions. This ensured a direct comparison of performance using the same combinations of prompt type and textual context. The experimental design systematically crossed three prompt formats: zero-shot, few-shot with three examples, and few-shot with nine examples, with three textual context conditions (none, sparse, detailed), yielding a total of nine configurations for each model.

4.3 Fine-tuning Setup

The fine-tuning experiments in this thesis were conducted to find out if targeted model adaptation could lessen a vision-language model's dependence on detailed text when answering perception questions about UML sequence diagrams. Fine-tuning was performed only on the Gemma 3 4B model. Its smaller size made it feasible to run multiple training iterations within the available resources, while also providing the opportunity to narrow the performance gap with larger models.

The fine-tuning process used the Axolotl framework [144], an open-source tool used to streamline post-training for AI models. It offers an easy environment for preparing datasets, setting up LoRA adapters, and managing training runs. Low-Rank

Adaptation (LoRA), as mentioned in Section 2.3.4, was chosen as the parameter-efficient fine-tuning method.

As noted in the previous section, three different fine-tuned models were trained. Each one used a dataset that matched one of the text context conditions. All training datasets were balanced across low, medium, and high complexity diagrams to avoid overfitting to any specific complexity level. Each training sample followed a consistent four-option multiple-choice format (A, B, C, D) with one clear correct answer. The evaluation set used for these models is described in Section 3.5.1, consisting of a total of 1,062 samples, which were used to train each of the fine-tuned variants.

4.3.1 Training Parameters Configuration

LoRA adapters were attached to both the language model layers and the vision-language projection layers of Gemma 3 4B, as detailed in Table 16. The configuration parameters varied slightly between the three fine-tuned variants:

1. **Model with no context:** rank $r = 16$, $\alpha = 16$, dropout = 0.05.
2. **Model with sparse context:** rank $r = 8$, $\alpha = 16$, dropout = 0.05.
3. **Model with detailed context:** rank $r = 8$, $\alpha = 16$, dropout = 0.05.

The explanation of these parameters is provided in the LoRA paragraph in Section 2.3.4. Only adapter parameters were updated during training, and these were merged into the base model weights after fine-tuning to produce a fully self-contained model for inference.

The general training parameters for all fine-tuning runs are summarized in Table 17. The sequence length was fixed at 1024 tokens after checking the tokenized input lengths in the training set. This length was enough to fit the largest sequence without cutting any part off. The number of epochs was set to three. This gave enough exposure to the dataset for effective learning while lowering the risk of overfitting. A weight decay of 0.01 was used for regularization. The learning rate of $1 \cdot 10^{-5}$ was chosen to ensure steady convergence without sudden changes in the loss curve. Training used the bfloat16 precision format, which retains the dynamic range of 32-bit floating point (FP32) while reducing memory usage and improving speed [145]. Sample packing was enabled to improve GPU use by combining shorter samples together within the fixed sequence length. This reduced padding and increased the number of effective tokens processed per batch [146].

Table 16: LoRA target layers in the language model and vision tower.

	Attention Layers	**MLP Layers**
Language Model	q_proj, k_proj, v_proj, o_proj	up_proj, down_proj, gate_proj
Vision Tower	q_proj, k_proj, v_proj, out_proj	fc1, fc2

Table 17: General training parameters used for fine-tuning runs.

Parameter	Value / Setting
Sequence length	1024 tokens
Micro-batch size	4
Gradient accumulation	none
Epochs	3
Optimizer	AdamW (fused)
Weight decay	0.01
Learning rate	$1 \cdot 10^{-5}$
Scheduler	Cosine decay
Warmup ratio	0.1
Precision	bfloat16
Sample packing	Enabled

4.3.2 Computational Environment for Fine-Tuning

The software environment and hardware specifications used for fine-tuning are documented in Tables 18 and 19, respectively.

Table 18: Software environment used for fine-tuning experiments.

Component	Version / Requirement
python	$\geq$ 3.11
axolotl	$\geq$ 0.10.0
transformers	$\geq$ 4.52.3
PEFT	$\geq$ 0.15.2

Table 19: Hardware specifications used for fine-tuning experiments.

Component	Specification
GPU	NVIDIA H100 Tensor Core GPU
GPU Memory	80 GB

4.4 Evaluation Prompts

For evaluation, two types of prompt designs were created: one for zero-shot and one for few-shot. The few-shot prompts followed a consistent format, differing only in the number of examples included, either three or nine, but were otherwise identical in structure. The prompt formats for both zero-shot and few-shot are shown in Figure 25.

Zero-Shot Prompt Format	Few-Shot Prompt Format
Role: User	**Role: User**
<Context + instruction>	<Context + Instruction of example>
<Question + answer options>	<Question + answer options of example>
<Image to be evaluated>	<Image of example>
	Role: Assistant
	<Correct answer>
	... *[repeated for 3/9 examples]*
	Role: User
	<Context + instruction>
	<Question + answer options>
	<Image to be evaluated>

Figure 25: Prompt design structure for zero-shot and few-shot evaluation settings.

For each zero-shot and few-shot setting, separate prompts were created for each combination of context condition: no context, sparse context, and detailed context. The prompt format was the same for all task types, but the questions differed depending on the task. All prompts included clear instructions to choose only the correct answer from four options. They were instructed to output only the letter (A, B, C, or D) of the selected answer. In prompts where additional context was provided, such as sparse or detailed, the contextual information was placed at the beginning of the prompt, followed by the question and answer choices. The instructions and overall template were identical for sparse and detailed contexts. They only varied in the content of the contextual information. The base instructions and prompt templates for the different types of context are shown in Figure 26.

4.5 Evaluation Framework

Each baseline and fine-tuned model was evaluated using the fixed evaluation split described in Section 3.5.2. For every test instance, the model's predicted answer was recorded and compared to the corresponding ground truth label. Performance was measured using the Exact Match (EM) metric, which looks at the proportion of responses that match the correct answer exactly. Since the model was instructed to output a single letter, EM scoring involved directly comparing the predicted letter to the expected one. Any outputs that did not follow this format, even if semantically correct, were counted as incorrect.

4.5.1 Evaluation Strategy

The evaluation procedure followed a consistent step-by-step strategy for all models and configurations.

No Context Prompt Template

Select the one correct answer and respond with only the letter corresponding to your choice. Do not include explanations or additional details.

`<Question + answer options>`

`<Image to be evaluated>`

Sparse/Detailed Context Prompt Template

Below is relevant information about the provided image. Use it to assist in answering the question given at the end.

`<Sparse/Detailed context>`

Select the one correct answer and respond with only the letter corresponding to your choice. Do not include explanations or additional details.

`<Question + answer options>`

`<Image to be evaluated>`

Figure 26: Prompt templates for no context and sparse/detailed context conditions for evaluations.

1. For each test question, the corresponding prompt format (zero-shot, three-shot, or nine-shot) was selected according to the experimental setting.
2. The appropriate textual context (none, sparse, or detailed) was inserted according to the evaluation condition.
3. The target UML diagram and the associated question were embedded in the designated position in the prompt template.
4. Each prompt was passed to the model using fixed inference parameters with deterministic decoding, also known as greedy sampling [147]. In this approach, the model selects the token with the highest probability at each step, ensuring that the same input always produces the same output. This eliminates randomness in generation and allows for consistent comparison across all models and evaluation runs.
5. The model output was limited to a single character (A, B, C, or D) to align with the multiple-choice question format.
6. The Exact Match metric was used to calculate the accuracy as the percentage of questions answered exactly correctly.

$$EMScore = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \cdot 100 \tag{1}$$

7. For robustness, the evaluation was repeated using five different seeds. The results were aggregated across seeds.

The recorded outputs made it possible to examine the five evaluation dimensions described in Section 4.1.

Table 20: Inference hyperparameters used during evaluation.

Parameter	Value
Number of Random seeds	5
Max model length	4,096 tokens (zero-shot and few-shot with 3 images)
	8,192 tokens (few-shot with 9 images)
Max output tokens	1
Temperature	0.0
Top-p	1.0

4.5.2 Inference Hyperparameters

All evaluations used the same inference hyperparameters to keep the results consistent and comparable between models, context types, and prompt formats. These settings enforce deterministic decoding and strict output to the multiple-choice format, eliminating stochastic variation in the generated answers. For zero-shot and few-shot (3-image) setups, a maximum model length of 4,096 tokens was enough to fit the largest prompt in this setting, including all images, textual context, and the task prompt. Using a larger limit would not have added any value and would have just increased GPU memory use and computation time. The few-shot setup with nine images required 8,192 tokens to include all content without risk of truncation. Table 20 lists the hyperparameters used for evaluation.

4.5.3 Computational Environment for Evaluation

All tasks were evaluated with the help of the vLLM framework. The software requirements for the evaluation of models are listed in Table 21. The hardware specifications were the same as those used for fine-tuning, already shown in Table 19.

Table 21: Software environment used for evaluation.

Component	Version / Requirement
python	$\geq$ 3.11
vLLM	$\geq$ 0.10.0
pillow	11.3.0

5 Results

This chapter presents the study's findings. Section 5.1 summarizes the main results and gives a detailed look at how contextual information, fine-tuning strategies, and few-shot prompting affect model performance. It also reviews how different models perform on the three types of tasks. Throughout this section, the results are supported by graphs and visuals that show performance trends, comparisons, and breakdowns by category. Section 5.2 discusses these findings in relation to the research questions introduced in Section 1.1.

5.1 Evaluation Outcomes

This section presents the evaluation results under different experimental conditions. It begins by analyzing the overall performance trends influenced by key factors such as textual context, few-shot prompting, and fine-tuning. These results give a broad picture of how each input setting affects model accuracy across tasks. Next, the section examines more specific details, analyzing how performance varies by question types and diagram complexity.

For a complete tabular overview of the evaluation results, refer to Appendix C.

5.1.1 Role of Context

The impact of textual context on the performance of multimodal models in perception tasks can be seen in Figure 27. It compares exact match accuracy across three context types: none, sparse, and detailed for Gemma 4B and Gemma 27B models. The results are further grouped by prompt format, including zero-shot, few-shot with three examples, and few-shot with nine examples.

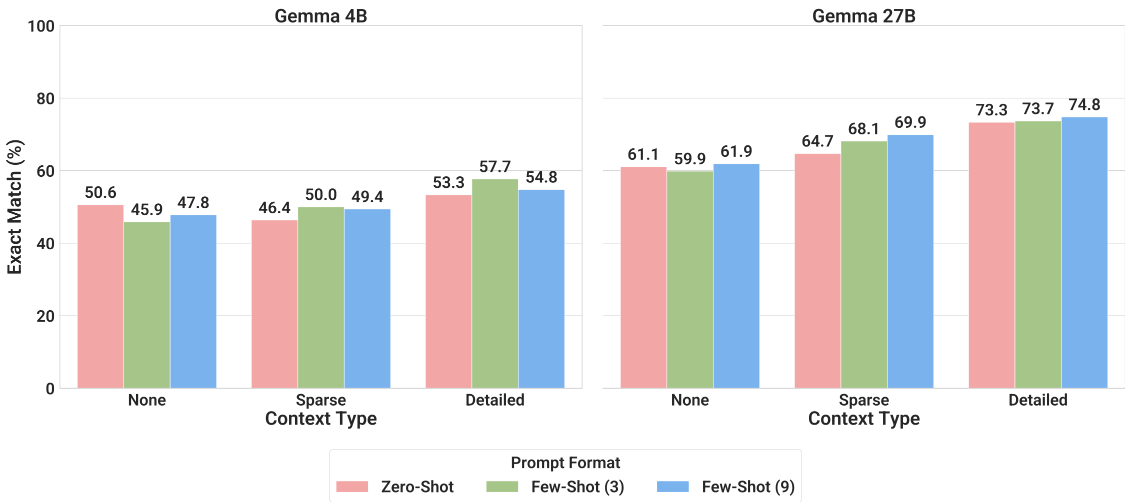


Figure 27: EM accuracy evaluated across context types (none, sparse, detailed) for each prompt format.

Overall, the results indicate that the addition of more textual context tends to improve the accuracy of the model, with detailed context consistently providing the best results for both models. Gemma 27B consistently outperforms Gemma 4B.

For Gemma 4B, the performance gains from context are modest. However, detailed context still provides a noticeable improvement. This indicates that smaller models can benefit from richer context, but with limited gains. In contrast, Gemma 27B shows a greater reliance on context. The model shows significant performance improvements from the absence of context to detailed context. This pattern remains consistent across both few-shot formats, highlighting that larger models are better at utilizing detailed textual cues. Performance increases linearly from no context to sparse context to detailed context, suggesting that the model progressively benefits from richer and more informative context.

5.1.2 Effect of Prompting Strategies

The effect of few-shot prompting on model performance across different context types is illustrated in Figure 28. For ease of readability, few-shot with three images is referred to as few-shot [3], and few-shot with nine images as few-shot [9].

For Gemma 4B, the impact of few-shot prompting is variable. While few-shot [3] yields the highest performance with detailed context, the improvement over zero-shot is minimal. Both few-shot formats perform similarly to the zero-shot setting, showing limited benefit from prompt examples. For Gemma 27B, the highest accuracy occurs with few-shot [9] in detailed context. However, the gains are small, similar to those of Gemma 4B. In fact, across all context types, the differences between zero-shot and few-shot prompting for Gemma 27B are minor. The larger models already perform well with zero-shot inputs and do not gain significantly from additional examples.

In summary, both models show limited gains from few-shot prompting. This highlights that contextual information matters more than prompt examples for these

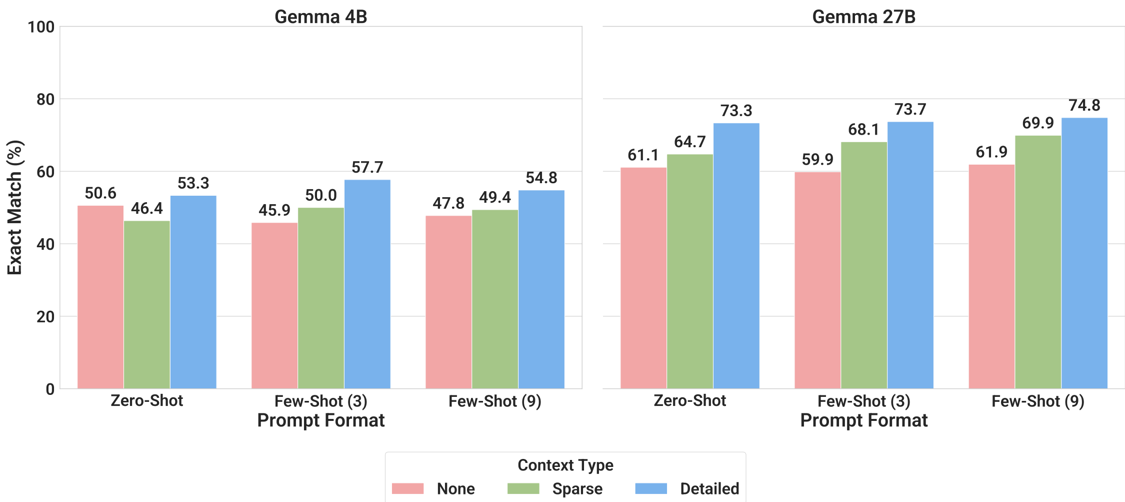


Figure 28: EM accuracy evaluated across prompting strategies (zero-shot, few-shot with 3 images, few-shot with 9 images) for each context type.

tasks, especially for the larger model.

5.1.3 Influence of Fine-Tuning

The Gemma 4B base model was fine-tuned using three different strategies. Each provided a different amount of text context during training. The objective is to determine whether fine-tuning can enhance model performance in perception tasks. The goal of fine-tuning is to reduce the model's dependence on prompt examples, allowing it to achieve strong performance in zero-shot scenarios. As a result, this section focuses mainly on zero-shot performance for all models. The exact match accuracy is evaluated for the base Gemma 4B model and its three fine-tuned variants, FT-N (fine-tuned with no context), FT-S (fine-tuned with sparse context), and FT-D (fine-tuned with detailed context). Figure 29 displays the zero-shot performance of the base Gemma 4B model, along with its fine-tuned versions, across three evaluation context types.

The results indicate that fine-tuning does not provide a substantial improvement over the base model. All models exhibit a similar pattern: performance declines from no context to sparse context, then increases with detailed context, where the highest scores are consistently achieved. The base model achieves the best overall results, but the differences between all models are small.

In addition to zero-shot evaluations, few-shot experiments were conducted using the fine-tuned Gemma 4B models. The results show that these models perform similarly to the base Gemma 4B model with all prompting strategies, including zero-shot, few-shot [3], and few-shot [9]. Table 22 reports the deviation of exact-match (EM) accuracy for each fine-tuned model relative to the base model. Negative values indicate a decrease in accuracy, and positive values indicate an increase. The low deviation values indicate that performance is largely consistent across models, context types, and prompting

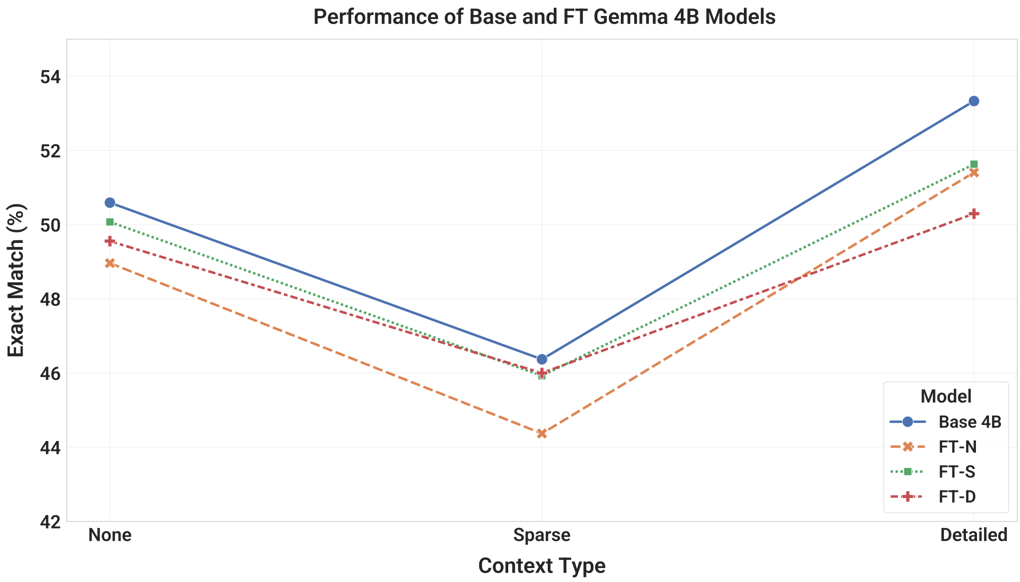


Figure 29: Zero-shot EM accuracy of Gemma 4B and its fine-tuned variants

Table 22: Deviation of none, sparse, and detailed context fine-tuned models from the base Gemma 4B model for each evaluation context under zero-shot and few-shot prompting.

Model	Zero-Shot			Few-Shot [3]			Few-Shot [9]		
	None	Sparse	Detailed	None	Sparse	Detailed	None	Sparse	Detailed
FT-N	−1.630	−2.000	−1.926	−0.222	0.000	−0.667	0.000	1.852	−2.371
FT-S	−0.519	−0.444	−1.703	1.926	−1.111	0.444	0.518	2.741	−1.111
FT-D	−1.037	−0.370	−3.037	−1.333	−2.667	−0.297	−1.778	0.593	−0.074

strategies. The performance of the fine-tuned models is mixed; no single variant consistently outperforms the others.

5.1.4 Performance by Task Type

The experiments were carried out on the three different types of tasks introduced in Section 3.4. Up to this point, the results have reflected overall performance trends across all tasks. This section now examines the performance of each task type individually, providing clearer insight into where the models excel and where they face challenges.

The fine-tuned versions of Gemma 4B (FT-N, FT-S, FT-D) are not included in this breakdown. Their results are largely similar to the base Gemma 4B model and do not offer additional information for task-level evaluation.

Location Recognition: For the location recognition task, there are clear performance differences between the Gemma 4B and Gemma 27B models, as shown in Figure 30.

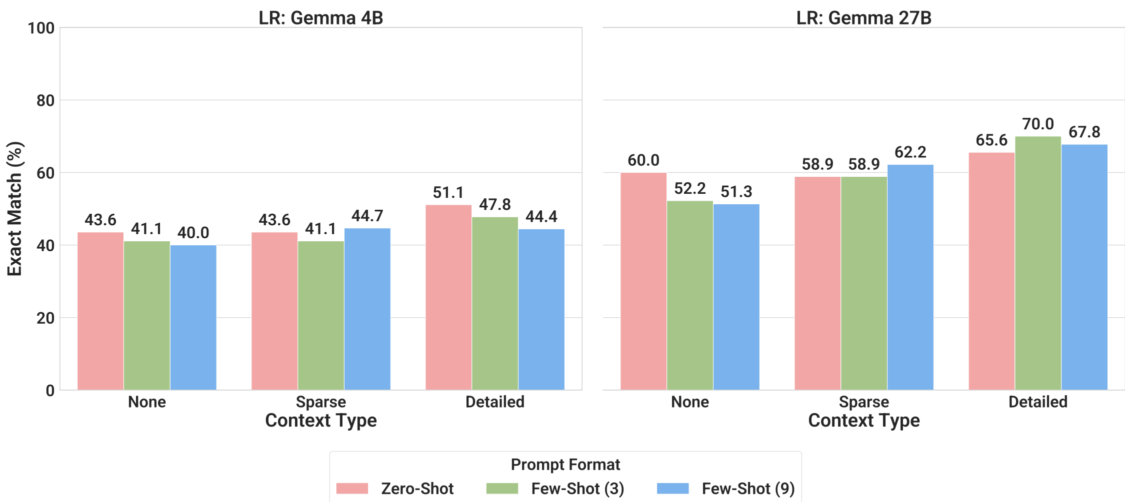


Figure 30: Performance of the location recognition task for Gemma 4B and 27B.

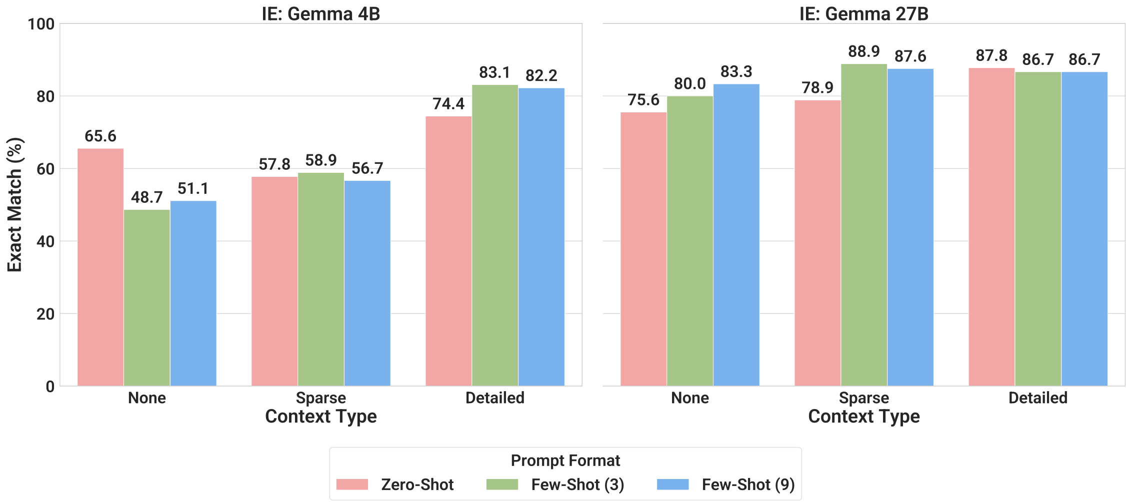


Figure 31: Performance of the information extraction task for Gemma 4B and 27B.

In terms of context, the results for Gemma 4B indicate that context provides only minimal improvements, with performance being quite similar for the none and sparse settings. For Gemma 27B, there is a clear improvement when detailed context is provided.

An important observation is that performance for no context actually decreases in the few-shot settings, which is particularly evident for Gemma 27B. For Gemma 4B, few-shot prompting also reduces performance when detailed context is used. In other cases, the gains from few-shot prompting are marginal, which suggests that extra examples do not consistently improve model accuracy for this task.

Information Extraction: The results for the information extraction task can be seen in Figure 31. Gemma 4B shows a substantial performance gain with detailed context, highlighting that contextual information is critical for this task. In contrast, Gemma 27B demonstrates strong baseline performance even without context. Adding more context does improve performance, but the relative gains are less significant than those seen with Gemma 4B.

Interestingly, few-shot prompting with no context significantly degrades the performance of Gemma 4B. Sparse context also shows only modest results, even in zero-shot conditions. For Gemma 27B, few-shot prompting offers only slight improvements, and the difference between providing three versus nine examples is negligible. Overall, these results show that Gemma 27B is very effective for information extraction across all prompt setups. In contrast, Gemma 4B needs richer contextual information to achieve competitive performance.

Sequence Flow Analysis: For the sequence flow analysis task, the performance trends of the Gemma models are shown in Figure 32. The results for Gemma 4B show a decline in performance with the addition of context for this task. This suggests that the extra information may not help the smaller model and might even be distracting.

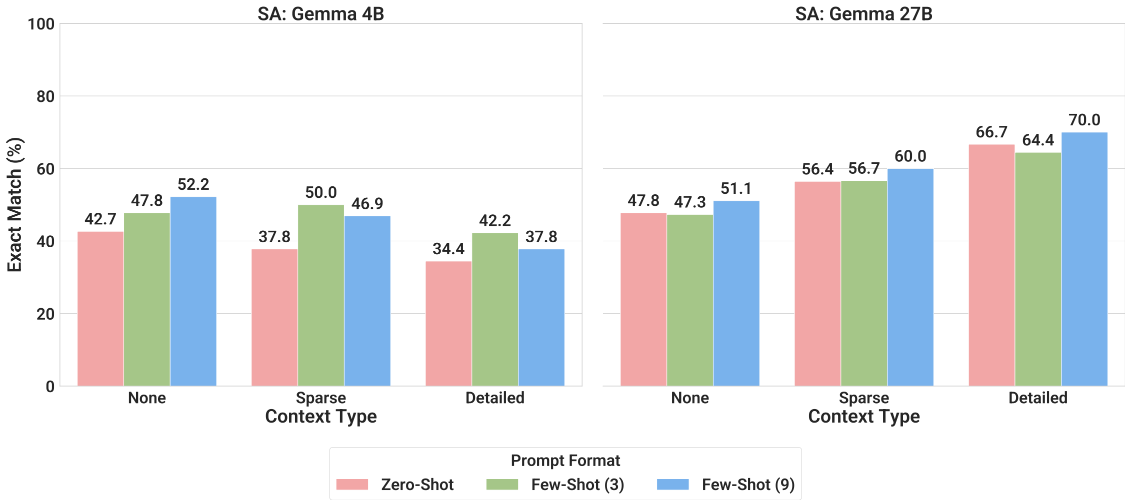


Figure 32: Performance of the sequence flow analysis task for Gemma 4B and 27B

In contrast, Gemma 27B improves with both sparse and detailed context, with detailed context achieving the best performance. Notably, without any context, Gemma 4B and Gemma 27B perform almost identically, indicating that this task relies primarily on the visual component of the model rather than the language component, as both models share the same vision encoder. However, with context included, Gemma 27B can use its larger parameter capacity to reach much higher accuracy.

For Gemma 4B, few-shot performance varies, with the best average results usually seen with three-shot prompts. For Gemma 27B, nine-shot prompting consistently beats three-shot prompting, although the improvements are slight. Overall, the difference between zero-shot and few-shot prompting is relatively small compared to the impact of context.

Comparison of Task Difficulties: The analysis presented for each task reveals differences in the levels of difficulty associated with the three task categories. Information Extraction (IE) is the most context-sensitive task, as demonstrated by Gemma 4B's performance, which declines significantly in the absence of rich contextual information. In contrast, Location Recognition (LR) presents a moderate level of difficulty. Both models demonstrate relatively stable performance under varying conditions. Sequence Flow Analysis (SA) stands out as the most difficult task. The performance of Gemma 4B decreases when additional context is introduced, suggesting that this smaller model may become distracted by extraneous textual information. In contrast, Gemma 27B appears adept at effectively utilizing the provided context.

5.1.5 Complexity-Wise Performance

Each question type in our evaluation dataset included images of varying complexities, categorized as *low*, *medium*, and *high*. In this section, the performance of models is analyzed across these complexity levels, focusing on how prompting and context richness affect accuracy for each task.

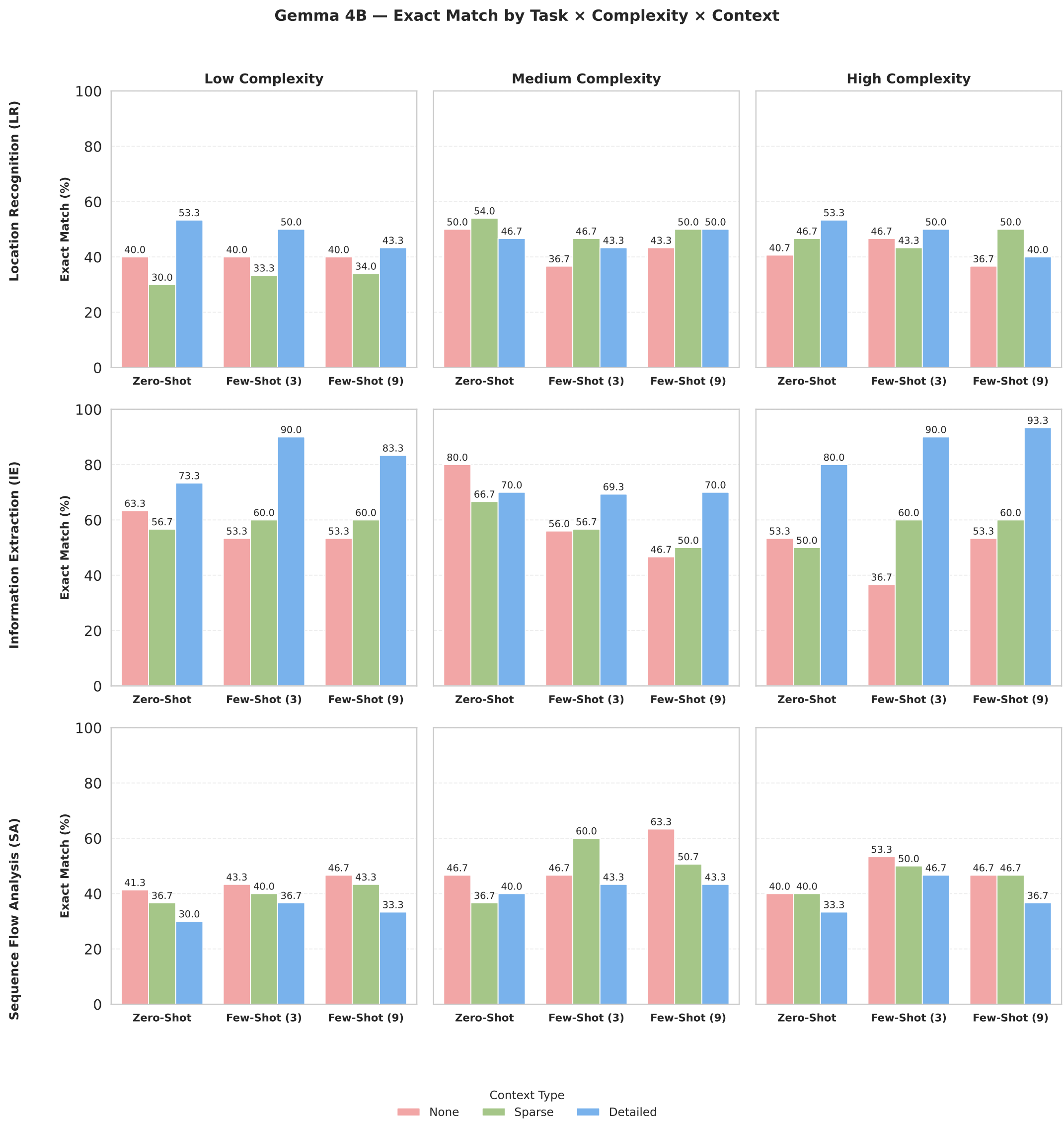


Figure 33: Complexity-wise performance of Gemma 4B across different prompt-context configurations for three different tasks: location recognition, information extraction, and sequence flow analysis.

Gemma 4B: Performance Trends by Complexity Level: Gemma 4B shows distinct trends in handling images of varying complexity as show in Figure 33.

Breaking down the results by task types gives more insight into how image complexity impacts performance:

1. **Location Recognition:** Low complexity images perform poorly in sparse context, which reduces accuracy no matter the prompting strategy. Performance improves significantly with detailed context. In contrast, medium and high complexity images show more consistent performance, with medium complexity images achieving the highest accuracy in all configurations.

2. **Information Extraction:** Medium complexity images perform best in zero-shot settings but see a decline with few-shot prompting. At all complexity levels, adding textual context consistently improves performance. High complexity images show the most significant improvements when switching from no context to detailed context.

3. **Sequence Flow Analysis:** All complexity levels usually perform better with few-shot prompting. The performance for few-shot [9] with no or sparse context is comparable to few-shot [3], but it drops sharply with detailed context in the three-image setting. Overall, medium-complexity images achieve the highest accuracy with few-shot prompts, although their performance also drops when detailed context is added.

Gemma 27B: Performance Trends by Complexity Level: The results for the performance of Gemma 27B on the three different complexities can be seen in Figure 34. The breakdown by task type is as follows:

1. **Location Recognition:** Context consistently benefits medium and high complexity images across all prompt configurations. Low complexity images, however, experience a performance drop with sparse context. Medium complexity achieves its best results under zero-shot prompting, while high complexity gains the most from few-shot prompting with detailed context.

2. **Information Extraction:** High complexity images struggle without context, so detailed context is essential for achieving strong results. Medium and low complexity images gain from context in zero-shot prompting, but their performance varies more when context is added in few-shot settings. Few-shot prompting helps to a degree, though increasing the number of examples yields diminishing returns. Low-complexity images with nine-shot prompting can still reach up to 96% accuracy.

3. **Sequence Flow Analysis:** Medium complexity images provide the best overall performance, especially in few-shot settings, compared to low and high complexity images. However, detailed context consistently leads to a noticeable drop in performance. This dip occurs in almost all prompt configurations and complexity levels for this task.

5.2 Discussion

This section summarizes the experimental findings and interprets them in relation to the study's research questions. Each section below restates the corresponding research question and presents a focused discussion based on the results and analyses from Section 5.1.

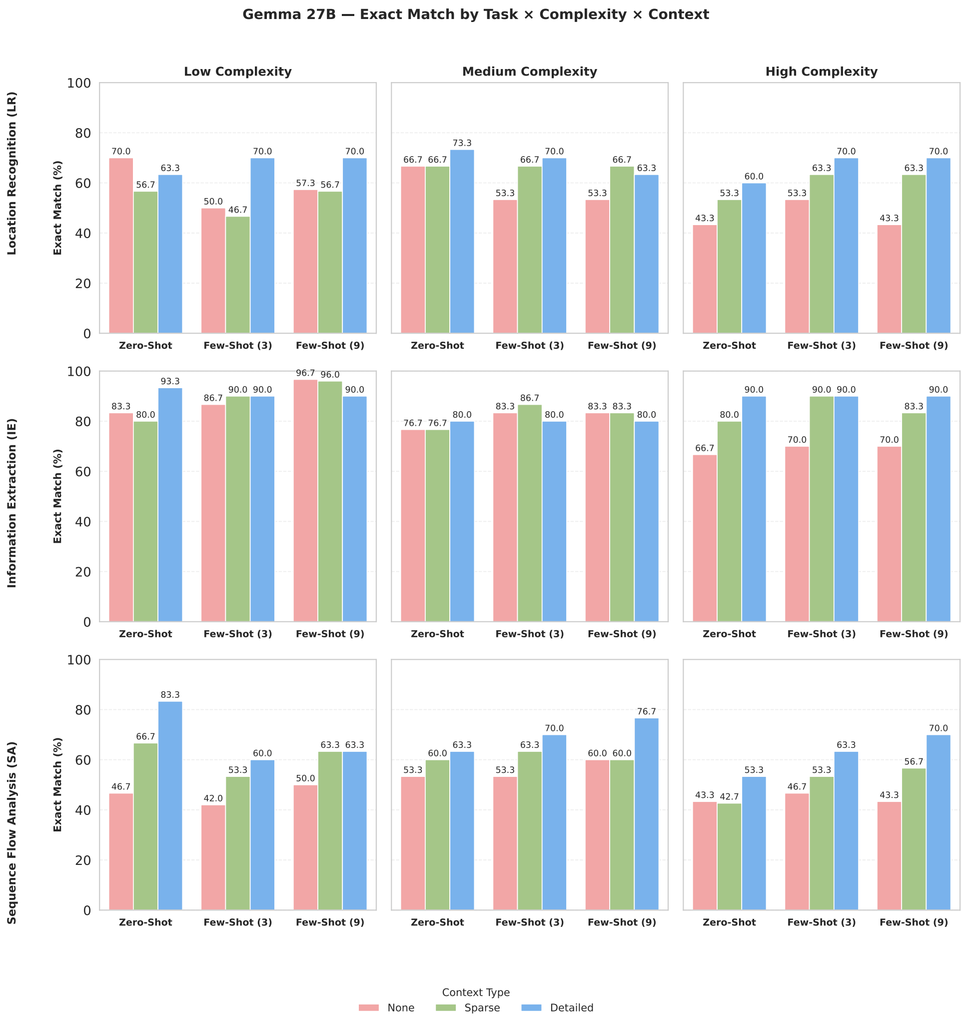


Figure 34: Complexity-wise performance of Gemma 27B across different prompt-context configurations for three tasks: location recognition, information extraction, and sequence flow analysis.

5.2.1 Addressing RQ1: Effect of Context

How does textual context affect the performance of vision language models when performing VQA on UML sequence diagrams?

Overall, the results show that adding more textual context generally improves model accuracy. Detailed context consistently gives the best performance for both Gemma 4B and Gemma 27B. Gemma 27B relies more on textual context, showing significant improvements as one moves from no context to sparse and then to detailed context, while Gemma 4B experiences more modest improvements. This suggests that larger models make better use of detailed text to improve VQA performance.

For Gemma 27B, performance increases roughly in a straight line as the context shifts from none to sparse to detailed. Gemma 4B mainly benefits from detailed context and shows little difference between no context and sparse context. Across task types, information extraction shows the most consistent gains from context.

5.2.2 Addressing RQ2: Impact of Few-shot Prompts

Does few-shot prompting improve model performance on basic perception tasks compared to zero-shot prompting?

Both Gemma 4B and Gemma 27B show limited benefits from few-shot prompting when compared to zero-shot performance. For Gemma 4B, few-shot [3] achieves slightly higher accuracy with detailed context, but the improvement over zero-shot is minimal. Few-shot [9] provides no consistent advantage and sometimes performs similarly or worse, especially without context. Gemma 27B follows a similar pattern: while the highest accuracy occurs with few-shot [9] in detailed context, the gains over zero-shot are small. In fact, the larger model already performs well in zero-shot settings, and more examples do not significantly improve its accuracy.

Overall, these findings suggest that contextual information is more important for performance than few-shot examples in these VQA tasks. Giving prompt examples without enough context can even lower accuracy, particularly for Gemma 27B in tasks like location recognition. The difference between three-shot and nine-shot prompting is small in most cases, showing diminishing returns from additional examples.

5.2.3 Addressing RQ3: Effectiveness of Fine-tuning

Can fine-tuning reduce a model's dependence on textual context, or is high-quality context still necessary?

Overall, fine-tuning with the LoRA adapter does not lead to significant performance gains in the performed experiments. All fine-tuned models of Gemma 4B show patterns similar to the base model. None of the fine-tuned versions consistently outperforms the base model, and the differences in exact-match accuracy across context types are minor. Few-shot tests with the fine-tuned models demonstrate similarly limited benefits. These results suggest that the fine-tuning recipes used in this thesis do not significantly lessen a model's reliance on textual context. High-quality contextual information is still essential for achieving strong performance, especially for detailed perception tasks in UML sequence diagrams.

However, these results should not be interpreted as evidence that fine-tuning is inherently ineffective. Rather, they suggest that the fine-tuning strategies applied here may not fully unlock the model's potential. It remains an open question whether alternative approaches, such as full model fine-tuning, training with larger and more diverse datasets, or separately training the vision encoder, could lead to better results. Exploring such approaches should be considered as an important direction for future work.

5.2.4 Addressing RQ4: Performance by Task Type

How does the impact of context, prompting, or fine-tuning on model performance vary depending on the type of task?

For location recognition, the influence of context and prompting is limited. Gemma 4B gets only slight gains from sparse or detailed context, while Gemma 27B shows more noticeable improvements when detailed context is available. Few-shot prompting often does not boost performance and can even slightly lower accuracy in situations without context.

Information extraction shows a strong dependence on textual context, especially for the smaller Gemma 4B model. Detailed context significantly improves accuracy, while performance drops in no context and sparse context situations. Gemma 27B achieves high baseline accuracy even without context but still benefits from the addition of context. Few-shot prompting provides limited additional gains, with little difference between 3-shot and 9-shot prompts.

Sequence flow analysis relies primarily on visual features, as both Gemma 4B and Gemma 27B perform similarly with no context, reflecting the shared vision encoder. For Gemma 4B, adding sparse or detailed context can sometimes harm performance, indicating that extra text may distract the smaller model. In contrast, Gemma 27B uses its larger capacity to improve steadily with more context, achieving the best results with detailed context. Once again, few-shot prompting has minimal effects.

5.2.5 Addressing RQ5: Complexity-Wise Trends

Do models struggle more with a particular complexity of images even when high-quality context is available?

Across both Gemma 4B and Gemma 27B, medium complexity images delivered the highest overall accuracy, particularly in location recognition and sequence flow analysis. High complexity images showed the greatest improvements from detailed context, especially in information extraction, where additional information helps the models understand relationships between multiple entities. However, their performance remained inconsistent. Conversely, low-complexity images exhibited surprisingly inconsistent results with accuracy often dropping for few-shot prompts. This might happen because simplified visuals offer too few explicit cues for reliable inference.

Overall, the results suggest that while detailed context helps address some challenges from extreme image complexities, it does not completely remove performance variability. Models show their greatest consistency on medium-complexity diagrams, with both low and high complexity images cause variability in performance.

6 Conclusion

This thesis investigated how textual context, few-shot prompting, and fine-tuning affect the performance of VLMs when used for perception tasks with UML sequence diagrams. The experiments revealed that model performance strongly relies on textual context, with larger models benefiting the most from high-quality contextual information. In contrast, few-shot prompting and fine-tuning did not significantly impact performance. The results also varied between the three types of tasks and the different levels of complexity of the diagram.

Section 6.1 presents the key takeaways from this study and summarizes the main experimental findings. Section 6.2 addresses the limitations that affect the interpretation of these results. Section 6.3 highlights the next directions for future research.

6.1 Key Takeaways

The findings of this thesis highlight several key insights into how VLMs handle VQA tasks on UML sequence diagrams. The most important of these is that textual context is the biggest factor affecting model performance. Both Gemma 4B and Gemma 27B showed improvement in results from richer textual context. Gemma 27B demonstrated an almost linear improvement as the input changed from none to sparse to detailed context, showing that a larger model can use text more effectively to improve accuracy.

Although providing textual context improved accuracy, including examples through few-shot prompting offered only limited benefits. Three-shot prompting occasionally improved results when combined with detailed context, but nine-shot prompting rarely produced further gains, and, in some situations, reduced performance. These findings suggest that examples alone cannot make up for the lack of meaningful textual context.

The experiments showed that fine-tuning with the applied LoRA configurations did not significantly reduce dependence on context. The fine-tuned versions of Gemma 4B produced results similar to the base model, with only minor fluctuations in accuracy in different contexts.

Performance also varied between task types. Information extraction was the most context-dependent task. Gemma 4B showed significant gains only when detailed text was provided, while Gemma 27B already had a strong baseline performance without additional context. Location recognition was comparatively less influenced by context or prompting, with a stable accuracy across conditions. Sequence flow analysis depended mostly on visual features, and so both models performed similarly without context, as they shared the same vision encoder. However, Gemma 27B used its larger language model to make small improvements when detailed context was available.

In terms of diagram complexity, medium-complexity diagrams produced the most consistent results across tasks and models. High-complexity diagrams improved significantly with detailed context, especially in information extraction, but performance still varied overall. Low-complexity diagrams showed surprising instability, with accuracy sometimes dropping during few-shot prompting

6.2 Limitations

Several limitations shaped the scope and interpretation of this thesis. A major limitation was the lack of publicly available datasets of UML sequence diagrams. The dataset used in this study was created from private documents and required extensive manual preprocessing, annotation, and vetting. As this effort was handled by a single researcher, there is a risk of human error in data curation, annotation consistency, and/or quality control.

Additionally, the choice of models was limited to open-source architectures because of the nature of the work and the need for reproducibility. Practical considerations, such as compatibility with inference and fine-tuning frameworks, along with the ability to run experiments within available resources, reduced the number of models that could be tested. As a result, the findings may not apply to other architectures that use different vision language fusion or pretraining strategies.

Furthermore, the experimental setup limited the length of the model responses to a single token. This choice was made to simplify the evaluation since the resources for large-scale human evaluation or for fine-tuning a dedicated judge language model were not available. Thus, the evaluation relied on the exact match metric. Although EM provides a clear and objective measure of correctness, it does not account for partial correctness, semantic similarity, or the reasoning behind the model outputs.

Fine-tuning was limited to LoRA-based training. Due to time and computational constraints, it was not possible to explore full-model fine-tuning, large-scale training, or targeted fine-tuning of the vision encoder. As a result, the conclusions about the limited effectiveness of fine-tuning in this thesis relate only to the particular recipes tested and should not be seen as proof against the wider potential of fine-tuning strategies.

6.3 Future Direction

Future research can build on this thesis in several meaningful directions. One area for improvement concerns the scale and diversity of data. The dataset used in this study was manually curated and limited in scope, which restricted both its coverage and generalizability. Expanding the task beyond sequence diagrams to include other UML types, such as class, activity, or state diagrams, or even unrelated structured visuals, would help assess the generalizability of the current findings. Additionally, evaluating other perception tasks that do not rely on textual information could provide further insight into the model's visual reasoning abilities.

The range of models evaluated should also be broadened. This thesis focused exclusively on Gemma models, but the variety of VLMs includes many different architectures, fusion mechanisms, pretraining goals, vision encoders, and language backbones. Evaluating a more diverse set of models would lead to more robust conclusions and help confirm the consistency of observations across different design choices.

Another direction for future research involves exploring more advanced fine-tuning strategies. This study used LoRA-based parameter-efficient fine-tuning applied to

Gemma 4B, which did not produce significant improvements. Future work could investigate full-model fine-tuning, adaptation of the vision encoder or fusion layers, or multi-stage training pipelines. Increasing both the training budget and the size of the model could further improve performance. A limitation of this study was the mismatch between fine-tuning and inference sequence lengths: fine-tuning was conducted with a maximum input length of 1,024 tokens, while inference allowed for 4,096 tokens in zero-shot and few-shot settings, and 8,192 tokens in few-shot settings. This discrepancy may have limited the model's ability to take full advantage of the contextual information provided. Aligning these lengths in future training runs, along with methods such as adaptive prompting, retrieval-augmented context, or curriculum learning based on diagram complexity, could enhance the effectiveness of fine-tuning.

The evaluation framework used in this thesis could also be expanded. While the exact match provided an efficient way to measure performance, it did not capture partial correctness, semantic similarity, or reasoning depth. Allowing longer outputs and incorporating evaluation methods such as human annotation or using language models as a judge to score would enable a more nuanced assessment of model capabilities.

References

[1] A. Vaswani *et al.*, "Attention is all you need", in *31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010, ISBN: 9781510860964.

[2] J. S. Ryu *et al.*, "Vision-language foundation models for medical imaging: A review of current practices and innovations", *Biomedical Engineering Letters*, vol. 15, no. 5, pp. 809–830, 2025. DOI: 10.1007/s13534-025-00484-6.

[3] X. Han *et al.*, "Multimodal fusion and vision-language models: A survey for robot vision", *CoRR*, vol. abs/2504.02477, Apr. 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.02477.

[4] Y. Zhou and N. Shimada, " Vision + Language Applications: A Survey ", in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2023, pp. 826–842. DOI: 10.1109/CVPRW59228.2023.00090. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPRW59228.2023.00090.

[5] P. Rahmanzadehgervi, L. Bolton, M. R. Taesiri, and A. T. Nguyen, "Vision language models are blind", in *Computer Vision – ACCV 2024: 17th Asian Conference on Computer Vision, Hanoi, Vietnam, December 8–12, 2024, Proceedings, Part V*, Hanoi, Vietnam: Springer-Verlag, 2024, pp. 293–309, ISBN: 978-981-96-0916-1. DOI: 10.1007/978-981-96-0917-8_17. [Online]. Available: https://doi.org/10.1007/978-981-96-0917-8_17.

[6] T. B. Brown *et al.*, "Language models are few-shot learners", in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[7] J. Wei *et al.*, "Finetuned language models are zero-shot learners", in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=gEZrGCozdqR.

[8] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning", in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=w0H2xGHlkw.

[9] L. Chen *et al.*, "Are we on the right way for evaluating large vision-language models?", in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: https://openreview.net/forum?id=evP9mxNNxJ.

[10] Z. Wang, B. Hooi, Y. Wang, M.-H. Yang, Z. Huang, and Y. Cai, "Text speaks louder than vision: ASCII art reveals textual biases in vision-language models", in *Second Conference on Language Modeling*, 2025. [Online]. Available: https://openreview.net/forum?id=naEyNVTLsh.

[11] P. Pezeshkpour, M. Aminnaseri, and E. Hruschka, *Mixed signals: Decoding vlms' reasoning and underlying bias in vision-language conflict*, 2025. arXiv: 2504.08974 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2504.08974.

[12] Z. Hu, J. Li, and Y. Yin, *When words outperform vision: VLMs can self-improve via text-only training for human-centered decision making*, 2025. arXiv: 2503.16965 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.16965.

[13] Object Management Group, *OMG Unified Modeling Language (OMG UML), Version 2.5*, https://www.uml-diagrams.org/sequence-diagrams.html, OMG Document Number: formal/2015-03-01, Published March 2015, 2015.

[14] G. Booch, *Object-oriented analysis and design with applications (2nd ed.)* USA: Benjamin-Cummings Publishing Co., Inc., 1993, ISBN: 0805353402.

[15] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*. USA: Addison Wesley Longman Publishing Co., Inc., 2004, ISBN: 0201403471.

[16] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented modeling and design*. USA: Prentice-Hall, Inc., 1991, ISBN: 0136298419.

[17] H. Störrle, "Describing process patterns with uml", in *Proceedings of the 8th European Workshop on Software Process Technology*, ser. EWSPT '01, Berlin, Heidelberg: Springer-Verlag, 2001, pp. 173–181, ISBN: 3540422641.

[18] J. Conallen, "Modeling web application architectures with UML", *Commun. ACM*, vol. 42, no. 10, pp. 63–70, Oct. 1999, ISSN: 0001-0782. DOI: 10.1145/317665.317677. [Online]. Available: https://doi.org/10.1145/317665.317677.

[19] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. USA: Addison Wesley Longman Publishing Co., Inc., 2004, ISBN: 0201775956.

[20] K. Goseva-Popstojanova *et al.*, "Architectural-level risk analysis using uml", *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 946–960, 2003. DOI: 10.1109/TSE.2003.1237174.

[21] H.-E. Eriksson and M. Penker, *Business Modeling With UML: Business Patterns at Work*, 1st. USA: John Wiley & Sons, Inc., 1998, ISBN: 0471295515.

[22] S. Cranefield and M. Purvis, "Uml as an ontology modelling language", ser. III'99, Stockholm, Sweden: CEUR-WS.org, 1999, pp. 44–51.

[23] D. Niyogi and S. N. Srihari, "A rule-based system for document understanding", in *AAAI Conference on Artificial Intelligence*, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:17495865.

[24] K. S. Mortensen and B. Belnap, "A rule-based approach employing feature recognition for engineering graphics characterisation", *Computer-Aided Engineering*, vol. 6, pp. 221–228, 1989. [Online]. Available: https://api.semanticscholar.org/CorpusID:57149084.

[25] N. Sadawi, "A rule-based approach for recognition of chemical structure diagrams", Ph.D. Thesis, University of Birmingham, 2013. [Online]. Available: http://etheses.bham.ac.uk/id/eprint/4325.

[26] A. Jain and R. Hoffman, "Evidence-based recognition of 3-d objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 6, pp. 783–802, 1988. DOI: 10.1109/34.9102.

[27] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984, ISBN: 0201055945.

[28] Y. Yu, A. Samal, and S. Seth, "A system for recognizing a large class of engineering drawings", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 8, pp. 868–890, 1997. DOI: 10.1109/34.608290.

[29] E. Abdelnabi, A. Maatuk, T. Abdelaziz, and S. Elakeili, "Generating uml class diagram using nlp techniques and heuristic rules", Dec. 2020. DOI: 10.1109/STA50679.2020.9329301.

[30] N. Otsu, "A threshold selection method from gray-level histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. DOI: 10.1109/TSMC.1979.4310076.

[31] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851.

[32] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. USA: Clarendon Press, 1986, ISBN: 0198539169.

[33] A. Okazaki, T. Kondo, K. Mori, S. Tsunekawa, and E. Kawamoto, "An automatic circuit diagram reader with loop-structure-based symbol recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 331–341, 1988. DOI: 10.1109/34.3898.

[34] E. De Jesus and R. De Alencar Lotufo, "Ecir-an electronic circuit diagram image recognizer", in *International Symposium on Computer Graphics, Image Processing, and Vision (Cat. No.98EX237)*, 1998, pp. 254–260. DOI: 10.1109/SIBGRA.1998.722758.

[35] A. Barta and I. Vajk, "Document image analysis by probabilistic network and circuit diagram extraction.", *Informatica (Slovenia)*, vol. 29, pp. 291–302, Oct. 2005.

[36] A. Barta and I. Vajk, "Document processing with bayesian network and agent-based programming", in *5th WSEAS International Conference on Signal, Speech and Image Processing*, WSEAS Press, Corfu, Greece, Aug. 2005, pp. 265–269.

[37] C. De Stefano, F. Tortorella, and M. Vento, "A method for the recognition of symbols on geographic maps", in *11th IAPR International Conference on Pattern Recognition*, 1992, pp. 734–737. DOI: 10.1109/ICPR.1992.201665.

[38] A. Mitiche and J. Aggarwal, "Image segmentation by conventional and information-integrating techniques: A synopsis", *Image and Vision Computing*, vol. 3, no. 2, pp. 50–62, 1985, ISSN: 0262-8856. DOI: https://doi.org/10.1016/0262-8856(85)90013-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0262885685900137.

[39] H. F. Ng, D. Jargalsaikhan, H.-C. Tsai, and C.-Y. Lin, "An improved method for image thresholding based on the valley-emphasis method", Oct. 2013, pp. 1–4. DOI: 10.1109/APSIPA.2013.6694261.

[40] Y. Liu and S. Srihari, "Document image binarization based on texture features", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 540–544, 1997. DOI: 10.1109/34.589217.

[41] W. Niblack, *An Introduction to Digital Image Processing*. Englewood Cliffs, N.J.: Prentice-Hall International, 1986, Bibliography: p. 207-210. Includes index. [Online]. Available: https://archive.org/details/introductiontodi0000nibl.

[42] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization", *Pattern Recognition*, vol. 33, no. 2, pp. 225–236, 2000, ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(99)00055-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320399000552.

[43] J. Bernsen, "Dynamic thresholding of gray level images", in *International Conference on Pattern Recognition*, Berlin, 1986, pp. 1251–1255.

[44] F. Chan, F. Lam, and H. Zhu, "Adaptive thresholding by variational method", *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 468–473, 1998. DOI: 10.1109/83.661196.

[45] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator", *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[46] J. M. S. Prewitt, "Object enhancement and extraction", in *Picture Processing and Psychopictorics*, B. Lipkin and A. Rosenfeld, Eds., New York: Academic Press, 1970, pp. 75–149.

[47] L. G. Roberts, "Machine perception of three-dimensional solids", Includes bibliographical references (leaf 81). Microfiche copy available in engineering., Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1963. [Online]. Available: http://hdl.handle.net/1721.1/11589.

[48] D. Marr and E. Hildreth, "Theory of edge detection", *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, vol. 207, pp. 187–217, Feb. 1980. DOI: 10.1098/rspb.1980.0020.

[49] P. Soille, *Morphological Image Analysis: Principles and Applications*, 2nd ed. Berlin, Heidelberg: Springer-Verlag, 2003, ISBN: 3540429883.

[50] Y. Vizilter, Y. Pyt'ev, A. Chulichkov, and L. M. Mestetskiy, "Morphological image analysis for computer vision applications", in *Computer Vision in Control Systems-1: Mathematical Theory*, M. N. Favorskaya and L. C. Jain, Eds. Cham: Springer International Publishing, 2015, pp. 9–58, ISBN: 978-3-319-10653-3. DOI: 10.1007/978-3-319-10653-3_2. [Online]. Available: https://doi.org/10.1007/978-3-319-10653-3_2.

[51] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected-component labeling for arbitrary image representations", *J. ACM*, vol. 39, pp. 253–280, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:1869184.

[52] N. R. Pal and S. K. Pal, "A review on image segmentation techniques", *Pattern Recognition*, vol. 26, no. 9, pp. 1277–1294, 1993, ISSN: 0031-3203.

[53] R. Szeliski, "Feature detection and matching", in *Computer Vision: Algorithms and Applications*. Springer International Publishing, 2022, pp. 333–399.

[54] M. Hassaballah, A. A. Abdelmgeid, and H. A. Alshazly, "Image features detection, description and matching", in *Image Feature Detectors and Descriptors : Foundations and Applications*, A. I. Awad and M. Hassaballah, Eds. Springer International Publishing, 2016, pp. 11–45. [Online]. Available: https://doi.org/10.1007/978-3-319-28854-3_2.

[55] C. G. Harris and M. J. Stephens, "A combined corner and edge detector", in *Alvey Vision Conference*, 1988. [Online]. Available: https://api.semanticscholar.org/CorpusID:1694378.

[56] D. Lowe, "Object recognition from local scale-invariant features", in *Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[57] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features", in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., ser. Lecture Notes in Computer Science, vol. 3951, Berlin, Heidelberg: Springer, 2006, pp. 404–417. DOI: 10.1007/11744023_32.

[58] A. Agarwal, D. Samaiya, and K. K. Gupta, "A comparative study of sift and surf algorithms under different object and background conditions", in *2017 International Conference on Information Technology (ICIT)*, 2017, pp. 42–45. DOI: 10.1109/ICIT.2017.48.

[59] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection", in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443, ISBN: 978-3-540-33833-8.

[60] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features", in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792, ISBN: 978-3-642-15561-1.

[61] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "ORB: An efficient alternative to SIFT or SURF.", in *ICCV*, D. N. Metaxas, L. Quan, A. Sanfeliu, and L. V. Gool, Eds., IEEE Computer Society, 2011, pp. 2564–2571, ISBN: 978-1-4577-1101-5. [Online]. Available: http://dblp.uni-trier.de/db/conf/iccv/iccv2011.html#RubleeRKB11.

[62] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions", in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, ser. SCG '04, Brooklyn, New York, USA: Association for Computing Machinery, 2004, pp. 253–262, ISBN: 1581138857. DOI: 10.1145/997817.997857. [Online]. Available: https://doi.org/10.1145/997817.997857.

[63] C. Cortes and V. Vapnik, "Support-vector networks", *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. DOI: 10.1007/BF00994018. [Online]. Available: http://dx.doi.org/10.1007/BF00994018.

[64] T. K. Ho, "Random decision forests", in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[65] T. Cover and P. Hart, "Nearest neighbor pattern classification", *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. DOI: 10.1109/TIT.1967.1053964.

[66] H. Zhang, "The optimality of naive bayes", in *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, Menlo Park, FL, USA: AAAI Press, May 2004, pp. 562–567.

[67] K. S. Refaat, W. N. Helmy, A. H. Ali, M. S. AbdelGhany, and A. F. Atiya, "A new approach for context-independent handwritten offline diagram recognition using support vector machines", in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 177–182. DOI: 10.1109/IJCNN.2008.4633786.

[68] M. D. and M. S., "Hand-drawn digital logic circuit component recognition using svm", *International Journal of Computer Applications*, vol. 143, pp. 24–28, Jun. 2016. DOI: 10.5120/ijca2016910058.

[69] S. Bernard, S. Adam, and L. Heutte, "Using random forests for handwritten digit recognition", in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, 2007, pp. 1043–1047. DOI: 10.1109/ICDAR.2007.4377074.

[70] P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana, "A review on probabilistic graphical models in evolutionary computation", *Journal of Heuristics*, vol. 18, no. 5, pp. 795–819, 2012. DOI: 10.1007/s10732-012-9208-4. [Online]. Available: https://doi.org/10.1007/s10732-012-9208-4.

[71] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data", in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289, ISBN: 1558607781.

[72] L. Rabiner and B. Juang, "An introduction to hidden markov models", *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986. DOI: 10.1109/MASSP.1986.1165342.

[73] Y. Qi, M. Szummer, and T. Minka, "Diagram structure recognition by bayesian conditional random fields", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, 191–196 vol. 2. DOI: 10.1109/CVPR.2005.129.

[74] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022. DOI: 10.1109/TNNLS.2021.3084827.

[75] B. Gosala, S. R. Chowdhuri, J. Singh, M. Gupta, and A. Mishra, "Automatic classification of UML class diagrams using deep learning technique: Convolutional neural network", *Applied Sciences*, vol. 11, no. 9, 2021, ISSN: 2076-3417. DOI: 10.3390/app11094267. [Online]. Available: https://www.mdpi.com/2076-3417/11/9/4267.

[76] X. Zhang, H. Washizaki, N. Yoshioka, and Y. Fukazawa, "Detecting design patterns in UML class diagram images using deep learning", in *2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2022, pp. 27–32. DOI: 10.1109/SNPD54884.2022.10051795.

[77] R. Zhu, W. Li, and C. Jin, "Tag: Uml activity diagram deeply supervised generation from business textural specification", in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2023, pp. 956–961. DOI: 10.1109/SANER56733.2023.00116.

[78] S. Minaee *et al.*, *Large language models: A survey*, 2025. arXiv: 2402.06196 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.06196.

[79] S. M. Lakew, M. Cettolo, and M. Federico, “A comparison of transformer and recurrent neural networks on multilingual neural machine translation.”, in *COLING*, E. M. Bender, L. Derczynski, and P. Isabelle, Eds., Association for Computational Linguistics, 2018, pp. 641–652. [Online]. Available: http://dblp.uni-trier.de/db/conf/coling/coling2018.html#LakewCF18.

[80] S. Karita *et al.*, “A comparative study on transformer vs rnn in speech applications”, in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, IEEE, Dec. 2019, pp. 449–456. DOI: 10.1109/asru46091.2019.9003750. [Online]. Available: http://dx.doi.org/10.1109/ASRU46091.2019.9003750.

[81] C. Sanford, D. Hsu, and M. Telgarsky, “Transformers, parallel computation, and logarithmic depth”, in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML’24, Vienna, Austria: JMLR.org, 2024.

[82] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. [Online]. Available: https://doi.org/10.18653/v1/N19-1423.

[83] Z. Liu, W. Lin, Y. Shi, and J. Zhao, “A robustly optimized bert pre-training approach with post-training”, in *Chinese Computational Linguistics: 20th China National Conference, CCL 2021, Hohhot, China, August 13–15, 2021, Proceedings*, Hohhot, China: Springer-Verlag, 2021, pp. 471–484, ISBN: 978-3-030-84185-0. DOI: 10.1007/978-3-030-84186-7_31. [Online]. Available: https://doi.org/10.1007/978-3-030-84186-7_31.

[84] A. Graves, *Generating sequences with recurrent neural networks*, 2014. arXiv: 1308.0850 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1308.0850.

[85] D. Naik, I. Naik, and N. Naik, “Decoder-only transformers: The brains behind generative ai, large language models and large multimodal models”, in *Contributions Presented at The International Conference on Computing, Communication, Cybersecurity and AI, July 3–4, 2024, London, UK*, N. Naik, P. Jenkins, S. Prajapat, and P. Grace, Eds., Cham: Springer Nature Switzerland, 2024, pp. 315–331, ISBN: 978-3-031-74443-3.

[86] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pretraining*, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, 2018.

[87] K. Aitken, V. V. Ramasesh, Y. Cao, and N. Maheswaranathan, “Understanding how encoder-decoder architectures attend”, in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, ser. NIPS ’21, Red Hook, NY, USA: Curran Associates Inc., 2021, ISBN: 9781713845393.

[88] C. Raffel *et al.*, “Exploring the limits of transfer learning with a unified text-to-text transformer”, *J. Mach. Learn. Res.*, vol. 21, no. 1, Jan. 2020, ISSN: 1532-4435.

[89] M. Lewis *et al.*, “BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”, in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. [Online]. Available: https://aclanthology.org/2020.acl-main.703/.

[90] X. Amatriain, *Prompt design and engineering: Introduction and advanced methods*, 2024. arXiv: 2401.14423 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2401.14423.

[91] Q. Dong *et al.*, “A survey on in-context learning”, in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 1107–1128. DOI: 10.18653/v1/2024.emnlp-main.64. [Online]. Available: https://aclanthology.org/2024.emnlp-main.64/.

[92] R. Agarwal *et al.*, “Many-shot in-context learning”, in *ICML 2024 Workshop on In-Context Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=goi7DFHlqS.

[93] J. Wei *et al.*, “Chain-of-thought prompting elicits reasoning in large language models”, in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS ’22, New Orleans, LA, USA: Curran Associates Inc., 2022, ISBN: 9781713871088.

[94] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, “Unleashing the potential of prompt engineering for large language models”, *Patterns*, vol. 6, no. 6, p. 101 260, Jun. 2025, ISSN: 2666-3899. DOI: 10.1016/j.patter.2025.101260. [Online]. Available: http://dx.doi.org/10.1016/j.patter.2025.101260.

[95] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, *Does prompt formatting have any impact on llm performance?*, 2024. arXiv: 2411.10541 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2411.10541.

[96] D. Machlab and R. Battle, *Llm in-context recall is prompt dependent*, 2024. arXiv: 2404.08865 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2404.08865.

[97] W. Zhou, S. Zhang, H. Poon, and M. Chen, “Context-faithful prompting for large language models”, in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14 544–14 556.

DOI: 10.18653/v1/2023.findings-emnlp.968. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.968/.

[98] K. W. Church, Z. Chen, and Y. Ma, "Emerging trends: A gentle introduction to fine-tuning", *Natural Language Engineering*, vol. 27, no. 6, pp. 763–778, 2021. DOI: 10.1017/S1351324921000322.

[99] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey", *Transactions on Machine Learning Research*, 2024, ISSN: 2835-8856. [Online]. Available: https://openreview.net/forum?id=lIsCS8b6zj.

[100] N. Ding, Y. Qin, G. Yang, Z. Liu, M. Sun, and J. Tang, "Parameter-efficient fine-tuning of large-scale pre-trained language models", *Nature Machine Intelligence*, vol. 5, pp. 220–235, 2023. DOI: 10.1038/s42256-023-00626-4. [Online]. Available: https://doi.org/10.1038/s42256-023-00626-4.

[101] H. Liu *et al.*, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning", in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22, New Orleans, LA, USA: Curran Associates Inc., 2022, ISBN: 9781713871088.

[102] E. J. Hu *et al.*, "LoRA: Low-rank adaptation of large language models", in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9.

[103] M. Mosbach, T. Pimentel, S. Ravfogel, D. Klakow, and Y. Elazar, "Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation", in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 12 284–12 314. DOI: 10.18653/v1/2023.findings-acl.779. [Online]. Available: https://aclanthology.org/2023.findings-acl.779/.

[104] Q. Yin *et al.*, "Deeper insights without updates: The power of in-context learning over fine-tuning", in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 4138–4151. DOI: 10.18653/v1/2024.findings-emnlp.239. [Online]. Available: https://aclanthology.org/2024.findings-emnlp.239/.

[105] F. Bordes *et al.*, *An introduction to vision-language modeling*, 2024. arXiv: 2405.17247 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.17247.

[106] K. Cheng, W. Song, Z. Ma, W. Zhu, Z. Zhu, and J. Zhang, "Beyond generic: Enhancing image captioning with real-world knowledge using vision-language pre-training model", in *Proceedings of the 31st ACM International Conference on Multimedia*, ser. MM '23, Ottawa ON, Canada: Association for Computing Machinery, 2023, pp. 5038–5047, ISBN: 9798400701085. DOI: 10.1145/

3581783.3611987. [Online]. Available: https://doi.org/10.1145/3581783.3611987.

[107] M. S. Nacson *et al.*, *Docvlm: Make your vlm an efficient reader*, 2024. arXiv: 2412.08746 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2412.08746.

[108] N. Sinha, V. Jain, and A. Chadha, "Guiding vision-language model selection for visual question-answering across tasks, domains, and knowledge types", *CoRR*, vol. abs/2409.09269, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.09269.

[109] J. Zhang, J. Huang, S. Jin, and S. Lu, "Vision-language models for vision tasks: A survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 8, pp. 5625–5644, 2024. DOI: 10.1109/TPAMI.2024.3369699.

[110] A. Ghosh, A. Acharya, S. Saha, V. Jain, and A. Chadha, *Exploring the frontier of vision-language models: A survey of current methodologies and future directions*, 2025. arXiv: 2404.07214 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2404.07214.

[111] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy.

[112] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning", in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=w0H2xGHlkw.

[113] J. Lee *et al.*, *Efficient llama-3.2-vision by trimming cross-attended visual features*, 2025. arXiv: 2504.00557 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2504.00557.

[114] S. Li and H. Tang, *Multimodal alignment and fusion: A survey*, 2024. arXiv: 2411.17040 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2411.17040.

[115] A. Radford *et al.*, "Learning transferable visual models from natural language supervision", in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html.

[116] J.-B. Alayrac *et al.*, "Flamingo: A visual language model for few-shot learning", in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22, New Orleans, LA, USA: Curran Associates Inc., 2022, ISBN: 9781713871088.

[117] H. Hu, X. Wang, Y. Zhang, Q. Chen, and Q. Guan, "A comprehensive survey on contrastive learning", *Neurocomput.*, vol. 610, no. C, Dec. 2024, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2024.128645. [Online]. Available: https://doi.org/10.1016/j.neucom.2024.128645.

[118] V. Hondru, F.-A. Croitoru, S. Minaee, *et al.*, "Masked image modeling: A survey", *International Journal of Computer Vision*, 2025. DOI: 10.1007/s11263-025-02524-1. [Online]. Available: https://doi.org/10.1007/s11263-025-02524-1.

[119] Z. Wang, Y. Yin, and I. Ramakrishnan, "Enhancing image-text matching with adaptive feature aggregation", in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 8245–8249. DOI: 10.1109/ICASSP48485.2024.10446913.

[120] Z. Wang, J. Yu, A. W. Yu, Z. Dai, Y. Tsvetkov, and Y. Cao, "SimVLM: Simple visual language model pretraining with weak supervision", in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=GUrhfTuf_3.

[121] L. Ding, K. Shih, H. Wen, X. Li, and Q. Yang, "Cross-attention transformer-based visual-language fusion for multimodal image analysis", *International Journal of Applied Science*, vol. 8, no. 1, p. 27, Feb. 2025. DOI: 10.30560/ijas.v8n1p27. [Online]. Available: https://doi.org/.

[122] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015. arXiv: 1503.02531 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1503.02531.

[123] P. Agrawal *et al.*, *Pixtral 12b*, 2024. arXiv: 2410.07073 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2410.07073.

[124] A. Q. Jiang *et al.*, *Mistral 7b*, 2023. arXiv: 2310.06825 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.06825.

[125] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2023. arXiv: 1606.08415 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1606.08415.

[126] M. AI, *Mistral nemo*, https://mistral.ai/news/mistral-nemo, Accessed: 2025-07-12, Jul. 2024.

[127] G. Team *et al.*, *Gemma 3 technical report*, 2025. arXiv: 2503.19786 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.19786.

[128] *Gemini: A family of highly capable multimodal models*, 2025. arXiv: 2312.11805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2312.11805.

[129] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, "Sigmoid loss for language image pre-training", in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 11 941–11 952. DOI: 10.1109/ICCV51070.2023.01100.

[130] A. Radford *et al.*, "Learning transferable visual models from natural language supervision", in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html.

[131] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints", in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. [Online]. Available: https://openreview.net/forum?id=hmOwOZWzYE.

[132] B. Zhang and R. Sennrich, "Root mean square layer normalization", in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[133] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing", in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, E. Blanco and W. Lu, Eds., Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. DOI: 10.18653/v1/D18-2012. [Online]. Available: https://aclanthology.org/D18-2012/.

[134] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit", *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, Jun. 2017, ISSN: 0163-5964. DOI: 10.1145/3140659.3080246. [Online]. Available: https://doi.org/10.1145/3140659.3080246.

[135] S. Bai *et al.*, *Qwen2.5-vl technical report*, 2025. arXiv: 2502.13923 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2502.13923.

[136] B. Heo, S. Park, D. Han, and S. Yun, "Rotary position embedding fornbsp;vision transformer", in *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part X*, Milan, Italy: Springer-Verlag, 2024, pp. 289–305, ISBN: 978-3-031-72683-5. DOI: 10.1007/978-3-031-72684-2_17. [Online]. Available: https://doi.org/10.1007/978-3-031-72684-2_17.

[137] M. Kaneko and T. Baldwin, *A little leak will sink a great ship: Survey of transparency for large language models from start to finish*, 2024. arXiv: 2403.16139 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2403.16139.

[138] Adobe Systems Incorporated, *PDF 1.7 Specification (ISO 32000-1)*, https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf, Accessed: 2025-06-16, 2008.

[139] Apache Software Foundation, *Apache License 2.0*, https://www.apache.org/licenses/LICENSE-2.0, 2004.

[140] W. Kwon *et al.*, *Efficient memory management for large language model serving with pagedattention*, 2023. arXiv: 2309.06180 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2309.06180.

[141] R. W. Hamming, "Error detecting and error correcting codes", *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[142] Anthropic, *Claude 3.5 Sonnet model card addendum*, https://www.anthropic.com/index/claude-3-5-sonnet-model-card, Accessed: 2025-06-17, 2024.

[143] E. Zhang, R. Yao, H. Liu, J. Yu, and J. Wang, *First multi-dimensional evaluation of flowchart comprehension for multimodal large language models*, 2024. arXiv: 2406.10057 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2406.10057.

[144] Axolotl maintainers and contributors, *Axolotl: Post-training for ai models*, https://github.com/axolotl-ai-cloud/axolotl, Accessed: 2025-08-13, 2023.

[145] D. Kalamkar *et al.*, *A study of bfloat16 for deep learning training*, 2019. arXiv: 1905.12322 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1905.12322.

[146] S. Wang, G. Wang, Y. Wang, J. Li, E. Hovy, and C. Guo, *Packing analysis: Packing is more appropriate for large models or datasets in supervised fine-tuning*, 2024. arXiv: 2410.08081 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2410.08081.

[147] C. Shi *et al.*, "A thorough examination of decoding methods in the era of LLMs", in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 8601–8629. DOI: 10.18653/v1/2024.emnlp-main.489. [Online]. Available: https://aclanthology.org/2024.emnlp-main.489/.

A Representative Examples from the Dataset

A.1 Example of Images

Example 1

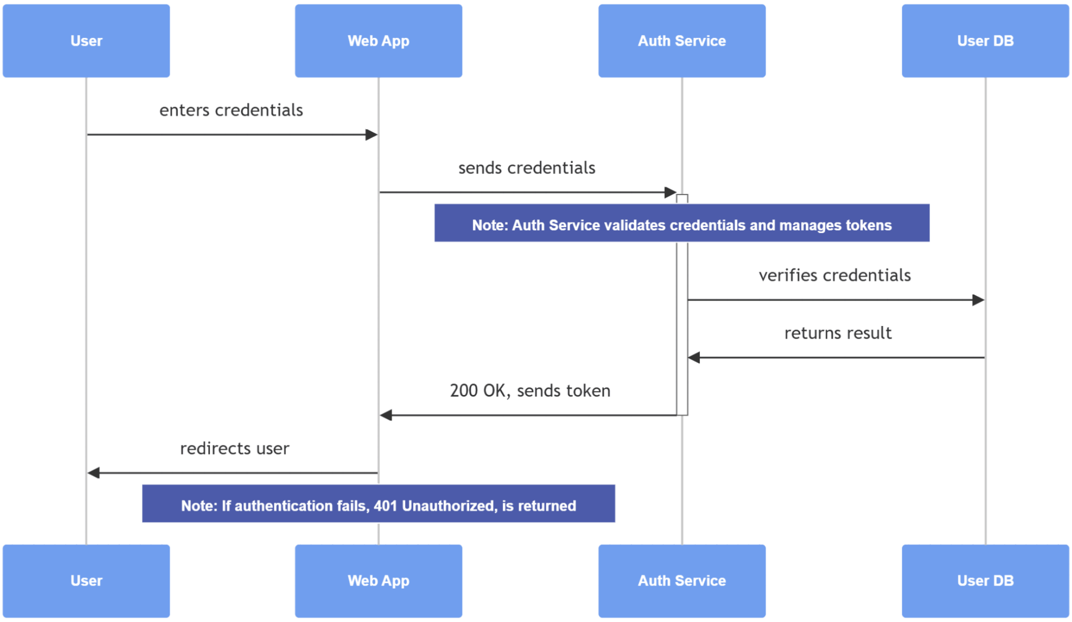


Figure A1: Example of a sequence diagram for user login and token issuance

Example 2

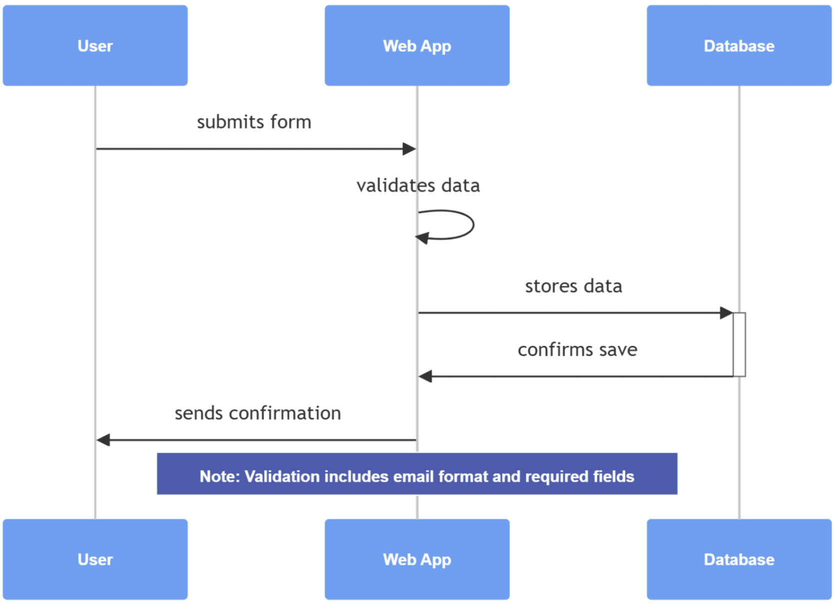


Figure A2: Example of a sequence diagram for form validation and data storage

A.2 Sparse and Detailed Context Examples

Context for Figure A1

Sparse Context:
Authentication flow diagram showing interactions between User, Web App, Auth Service, and User DB components. The Auth Service acts as a central authentication handler, validating credentials against the User DB and managing tokens. Successful authentication returns a 200 OK response with token, while failed attempts result in 401 Unauthorized errors.

Detailed Context:
Authentication flow diagram showing interactions between User, Web App, Auth Service, and User DB components. The Auth Service acts as a central authentication handler, validating credentials against the User DB and managing tokens. Successful authentication returns a 200 OK response with token, while failed attempts result in 401 Unauthorized errors.

Main Components:
- User: End user initiating authentication
- Web App: Frontend application interface
- Auth Service: Authentication service handling credentials and tokens
- User DB: Database storing user credentials and information

Sequence Flows:
#1. User enters credentials into Web App
#2. Web App sends credentials to Auth Service
#3. Auth Service verifies credentials with User DB
#4. User DB returns verification result to Auth Service
#5. Auth Service sends 200 OK response with token to Web App

Conditional Logic:
#1. Auth Service validates credentials and manages tokens
#2. If authentication fails, system returns 401 Unauthorized error

Other Observations:
- System implements standard authentication flow pattern
- Token-based authentication approach for session management
- Error handling included for failed authentication attempts

Context for Figure A2

Sparse Context:
Sequence diagram showing form submission workflow between User, Web App, and

Database components, with data validation and storage steps. Validation process checks email format and required field completion before storing data and returning confirmation.

Detailed Context:
Sequence diagram showing form submission workflow between User, Web App, and Database components, with data validation and storage steps. Validation process checks email format and required field completion before storing data and returning confirmation.

Main Components:
- User: End user interacting with the system
- Web App: Application interface handling form submission and validation
- Database: Storage system for persisting form data

Sequence Flows:
#1. User submits form to Web App
#2. Web App performs internal validation of the submitted data
#3. Web App stores validated data in Database
#4. Database confirms successful data storage with arrow to Web App
#5. Web App sends confirmation message back to User

Conditional Logic:
#1. Data validation check includes email format verification

Other Observations:
- The validation step is performed as a self-referential action within the Web App component
- The diagram uses standard sequence diagram notation with solid arrows for messages
- The interaction pattern follows a typical form submission workflow

A.3 Question/Answer Examples

Questions/Answers for Figure A1
Three example questions have been generated for each task category. Correct answers are indicated in green text.

Information Extraction:

1. What is the status code and action mentioned when authentication is successful?
   A) 200 OK, sends token
   B) 201 Created, sends token
   C) 200 OK, validates credentials
   D) 202 Accepted, sends token

2. According to the note in the diagram, what happens if authentication fails?

A) 403 Forbidden, is returned
B) 400 Bad Request, is returned
C) 401 Unauthorized, is returned
D) 404 Not Found, is returned

3. What is the primary function of the Auth Service according to the note in the diagram?
A) Manages users and validates tokens
B) Creates tokens and manages users
C) Authenticates users and creates databases
D) Validates credentials and manages tokens

Location Recognition:

1. Which component is positioned rightmost in the sequence diagram?
A) Web App
B) Auth Service
C) User
D) User DB

2. Where is the first note located in the diagram?
A) After the first message
B) After the third message
C) After the second message
D) After the fourth message

3. How many components are positioned between Web App and User DB?
A) 0
B) 1
C) 2
D) 3

Sequence Flow Analysis:

1. What is the first interaction flow in the authentication process?
A) Web App to Auth Service
B) User to Auth Service
C) User to Web App
D) Auth Service to User DB

2. When authentication is successful, what is the sequence of responses?
A) User DB to Auth Service to Web App
B) User DB to Web App to User
C) Auth Service to User DB to Web App
D) Web App to Auth Service to User

3. Which component is responsible for validating credentials and managing tokens?
   A) User DB
   B) Auth Service
   C) Web App
   D) All components jointly

Questions/Answers for Figure A2
Three example questions have been generated for each task category. Correct answers are indicated in green text.

Information Extraction:

1. What is the text shown in the note at the bottom of the diagram?
   A) Note: Email validation and required fields
   B) Note: Validation includes email format and required fields
   C) Note: Form validation and email requirements
   D) Note: Required fields and email validation format

2. What is the first action shown in the message flow?
   A) validates data
   B) stores data
   C) submits form
   D) sends confirmation

3. Which components are shown in the diagram?
   A) Client, Web App, Database
   B) User, Web App, Storage
   C) User, Web App, Database
   D) Browser, Application, Database

Location Recognition:

1. What is the position of the Web App component in relation to the other components?
   A) Leftmost
   B) Center
   C) Rightmost
   D) Below the Database

2. Which message flow immediately follows the "validates data" self-referential arrow?
   A) confirms save
   B) sends confirmation
   C) submits form
   D) stores data

3. What is the relative position of the validation note box in the diagram?
   A) Top of the diagram
   B) Middle of the diagram
   C) Bottom third of the diagram
   D) Between initial component boxes

Sequence Flow Analysis:

1. What is the first interaction in the sequence diagram?
   A) Web App validates data
   B) User submits form to Web App
   C) Database stores data
   D) Web App sends confirmation to User

2. After data validation, what is the next message flow?
   A) Web App to User
   B) Database to Web App
   C) Web App to Database
   D) User to Web App

3. Which component performs the validation of email format and required fields?
   A) Database
   B) User
   C) Web App
   D) External validator

B Prompts Used for Dataset Generation

B.1 Base Instruction for Image Complexity Classification

You are given a UML sequence diagram that illustrates interactions between system components over time. Your task is to classify the overall complexity of the diagram as low, medium or high. Follow these steps in order before deciding:

Step 1: Identify all the lifelines in the diagram.
- A lifeline represents an individual participant (such as a class, user, or system).
- It is shown as a rectangle (head) followed by a vertical dashed line.

Step 2: Count the number of lifelines.

Step 3: Identify all the messages.
- Messages are arrows showing communications between lifelines (e.g., method calls, responses, signals).
- Include synchronous, asynchronous, return, and self-messages.

Step 4: Count the number of messages in the diagram.

Step 5: Apply the following rules:
Low Complexity: 2 to 4 lifelines OR fewer than 8 messages
- Medium Complexity: 5 to 6 lifelines OR 8 to 15 messages
- High Complexity: 7 or more lifelines OR more than 15 messages
- Rule: If lifelines suggest one complexity class and messages suggest another, choose the higher one.

At the end, return only one word as your final answer: low, medium or high.

B.2 Prompts for Question/Answer Generation

B.2.1 Information Extraction

You are an AI assistant specialized in analyzing message sequence diagrams and generating multiple-choice questions about textual information within them. Given the message sequence diagram and the accompanying context, your task is to generate three multiple-choice questions that evaluate the ability to extract specific textual content visible in the diagram.

DIAGRAM CONTEXT: {context}

Rules:
1. Focus only on extracting visible textual content from the diagram. Do not interpret message flows or relationships.

2. Each question must ask about identifying specific text, labels, or names that appear in the diagram.
3. Provide exactly four answer options per question, with only one correct answer.
4. Include the correct answers at the end of the output, but do not indicate correctness within the options.
5. Do not use any information from the provided context when forming questions or answers.
6. Follow the template strictly and provide no additional commentary.

Example Question Types:
1. What is the text content in the [color/shape] box?
2. What condition is specified in the `alt` or `opt` fragment?
3. What components are listed inside the boundary box?
4. What information is contained in the note attached to [element]?
5. Which of these terms appears in the [specific area] of the diagram?
6. What options are listed in the decision box?
7. What is the title or heading of the diagram?
8. What textual information is shown inside [specific region]?

Template:
#Q1#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]

#Q2#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]

#Q3#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]

#Correct Answers#: [Ans1], [Ans2], [Ans3]

B.2.2 Location Recognition

You are an AI assistant specialized in analyzing message sequence diagrams and generating multiple-choice questions about spatial relationships within them. Given the message sequence diagram and the accompanying context, your task is to generate three multiple-choice questions that evaluate understanding of positional relationships between entities or message flows.

DIAGRAM CONTEXT: {context}

Rules:
1. Focus only on the relative positions of systems, messages, arrows, or other structural elements in the diagram.
2. Each question must test spatial reasoning or ordering of elements in the diagram.
3. Provide exactly four plausible answer options per question, with only one correct answer.
4. Include the correct answers at the end of the output, but do not indicate correctness within the answer options.
5. Do not use any information from the provided context to form the questions or answers.
6. Follow the template strictly and provide no additional explanatory text.

Example Question Types:
1. What is the position of [element A] relative to [element B]?
2. Which components are contained in [boundary/region]?
3. What message or action immediately follows [specific action]?
4. What is the leftmost, rightmost, or second-from-left/right component in the diagram?
5. What is the topmost or bottommost arrow in the diagram?
6. Which system initiates the first message?
7. How many systems are positioned between [element A] and [element B]?
8. What step or message precedes [specific action]?

Template:

#Q1#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]

#Q2#: [Question text]
A) [Option A]

B) [Option B]
C) [Option C]
D) [Option D]

#Q3#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]

#Correct Answers#: [Ans1], [Ans2], [Ans3]

B.2.3 Sequence Flow Analysis

You are an AI assistant specialized in analyzing message sequence diagrams, with a focus on sequence flow, conditional logic, and arrow directions. Given the message sequence diagram and the accompanying context, your task is to generate three multiple-choice questions that evaluate understanding of these aspects.

DIAGRAM CONTEXT: {context}

Rules:
1. Focus only on who sends specific messages to whom and the direction of message flows.
2. Each question must correspond to exactly one of the following categories: a sequence flow (#S), a conditional logic element (#C), or a valid inference not tied directly to any specific step (#NA).
3. Base all questions on message flow, senders, receivers, or arrow direction.
4. Provide four plausible answer options per question, with only one correct answer.
5. Include the correct answers at the end of the output, but do not mark them within the options.
6. Follow the template exactly and provide no additional text outside the required format.
7. When specifying the Flow/Logic tag, use #S for sequence flows (e.g., #S4), #C for conditions (e.g., #C2), or #NA for inferences.
8. If using #NA, ensure the question relies on a reasonable inference but is not identical to a listed step or condition.

Example Question Types:
1. Which arrow represents the [Message X] flow?
2. What is the direction of the [Message Y] arrow?
3. Which component sends [Message Z] to which component?

Template:

#Q1#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]
#Flow/Logic# [#S number or #C number or #NA]

#Q2#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]
#Flow/Logic# [#S number or #C number or #NA]

#Q3#: [Question text]
A) [Option A]
B) [Option B]
C) [Option C]
D) [Option D]
#Flow/Logic# [#S number or #C number or #NA]

#Correct Answers#: [Ans1], [Ans2], [Ans3]

C Complete Results

Ques. Type	Complexity	Zero Shot			Few-Shot (3 images)			Few-Shot (9 images)		
		None	Sparse	Detailed	None	Sparse	Detailed	None	Sparse	Detailed
LR	High	40.667	46.667	53.333	46.667	43.333	50.000	36.667	50.000	40.000
LR	Medium	50.000	54.000	46.667	36.667	46.667	43.333	43.333	50.000	50.000
LR	Low	40.000	30.000	53.333	40.000	33.333	50.000	40.000	34.000	43.333
LR	Overall	43.556	43.556	51.111	41.111	41.111	47.778	40.000	44.667	44.444
IE	High	53.333	50.000	80.000	36.667	60.000	90.000	53.333	60.000	93.333
IE	Medium	80.000	66.667	70.000	56.000	56.667	69.333	46.667	50.000	70.000
IE	Low	63.333	56.667	73.333	53.333	60.000	90.000	53.333	60.000	83.333
IE	Overall	65.556	57.778	74.444	48.667	58.889	83.111	51.111	56.667	82.222
SA	High	40.000	40.000	33.333	53.333	50.000	46.667	46.667	46.667	36.667
SA	Medium	46.667	36.667	40.000	46.667	60.000	43.333	63.333	50.667	43.333
SA	Low	41.333	36.667	30.000	43.333	40.000	36.667	46.667	43.333	33.333
SA	Overall	42.667	37.778	34.444	47.778	50.000	42.222	52.222	46.889	37.778
ALL	High	44.667	45.556	55.556	45.556	51.111	62.222	45.556	52.222	56.667
ALL	Medium	58.889	52.444	52.222	46.444	54.444	52.000	51.111	50.222	54.444
ALL	Low	48.222	41.111	52.222	45.556	44.444	58.889	46.667	45.778	53.333
ALL	Overall	50.593	46.370	53.333	45.852	50.000	57.704	47.778	49.407	54.815

Table C1: Performance (%) of Gemma 4B across question types and evaluation setups (Zero-shot and Few-shot).

Ques. Type	Complexity	Zero Shot			Few-Shot (3 images)			Few-Shot (9 images)		
		None	Sparse	Detailed	None	Sparse	Detailed	None	Sparse	Detailed
LR	High	43.333	53.333	60.000	53.333	63.333	70.000	43.333	63.333	70.000
LR	Medium	66.667	66.667	73.333	53.333	66.667	70.000	53.333	66.667	63.333
LR	Low	70.000	56.667	63.333	50.000	46.667	70.000	57.333	56.667	70.000
LR	Overall	60.000	58.889	65.556	52.222	58.889	70.000	51.333	62.222	67.778
IE	High	66.667	80.000	90.000	70.000	90.000	90.000	70.000	83.333	90.000
IE	Medium	76.667	76.667	80.000	83.333	86.667	80.000	83.333	83.333	80.000
IE	Low	83.333	80.000	93.333	86.667	90.000	90.000	96.667	96.000	90.000
IE	Overall	75.556	78.889	87.778	80.000	88.889	86.667	83.333	87.556	86.667
SA	High	43.333	42.667	53.333	46.667	53.333	63.333	43.333	56.667	70.000
SA	Medium	53.333	60.000	63.333	53.333	63.333	70.000	60.000	60.000	76.667
SA	Low	46.667	66.667	83.333	42.000	53.333	60.000	50.000	63.333	63.333
SA	Overall	47.778	56.444	66.667	47.333	56.667	64.444	51.111	60.000	70.000
ALL	High	51.111	58.667	67.778	56.667	68.889	74.444	52.222	67.778	76.667
ALL	Medium	65.556	67.778	72.222	63.333	72.222	73.333	65.556	70.000	73.333
ALL	Low	66.667	67.778	80.000	59.556	63.333	73.333	68.000	72.000	74.444
ALL	Overall	61.111	64.741	73.333	59.852	68.148	73.704	61.926	69.926	74.815

Table C2: Performance (%) of Gemma 27B across question types and evaluation setups (Zero-shot and Few-shot).

Ques. Type	Complexity	Zero Shot			Few-Shot (3 images)			Few-Shot (9 images)		
		None	Sparse	Detailed	None	Sparse	Detailed	None	Sparse	Detailed
LR	High	30.000	43.333	50.000	43.333	43.333	50.000	36.667	50.000	40.667
LR	Medium	50.000	53.333	46.000	34.000	43.333	46.667	43.333	50.000	50.000
LR	Low	34.000	36.667	50.000	40.000	33.333	46.667	43.333	37.333	34.000
LR	Overall	38.000	44.444	48.667	39.111	40.000	47.778	41.111	45.778	41.556
IE	High	53.333	50.000	80.000	36.667	60.000	90.000	53.333	60.000	90.000
IE	Medium	83.333	63.333	70.000	56.667	56.667	63.333	46.667	53.333	70.000
IE	Low	66.667	49.333	73.333	53.333	60.000	90.000	53.333	60.000	80.667
IE	Overall	67.778	54.222	74.444	48.889	58.889	81.111	51.111	57.778	80.222
SA	High	37.333	33.333	30.000	53.333	50.000	46.667	46.667	53.333	33.333
SA	Medium	46.000	36.667	33.333	50.000	60.000	43.333	60.000	53.333	40.000
SA	Low	40.000	33.333	30.000	43.333	43.333	36.667	46.667	44.000	33.333
SA	Overall	41.111	34.444	31.111	48.889	51.111	42.222	51.111	50.222	35.556
ALL	High	40.222	42.222	53.333	44.444	51.111	62.222	45.556	54.444	54.667
ALL	Medium	59.778	51.111	49.778	46.889	53.333	51.111	50.000	52.222	53.333
ALL	Low	46.889	39.778	51.111	45.556	45.556	57.778	47.778	47.111	49.333
ALL	Overall	48.963	44.370	51.407	45.630	50.000	57.037	47.778	51.259	52.444

Table C3: Performance (%) of Gemma 4B fine-tuned on no-context training data across question types and evaluation setups (Zero-shot and Few-shot).

Ques. Type	Complexity	Zero Shot			Few-Shot (3 images)			Few-Shot (9 images)		
		None	Sparse	Detailed	None	Sparse	Detailed	None	Sparse	Detailed
LR	High	33.333	46.000	53.333	43.333	43.333	50.000	40.000	50.000	40.000
LR	Medium	46.667	46.667	40.667	40.000	43.333	46.667	43.333	43.333	50.000
LR	Low	46.667	36.667	50.667	40.000	33.333	53.333	43.333	36.667	40.000
LR	Overall	42.222	43.111	48.222	41.111	40.000	50.000	42.222	43.333	43.333
IE	High	50.667	50.000	80.000	40.000	60.000	90.000	50.667	60.000	86.667
IE	Medium	83.333	66.667	70.000	60.000	60.000	63.333	46.667	56.667	70.000
IE	Low	66.667	60.000	76.667	56.667	56.667	86.667	50.667	60.000	80.000
IE	Overall	66.889	58.889	75.556	52.222	58.889	80.000	49.333	58.889	78.889
SA	High	36.667	37.333	30.000	56.667	46.667	46.667	46.667	52.667	36.667
SA	Medium	46.667	33.333	36.667	50.000	56.667	46.667	63.333	60.000	43.333
SA	Low	40.000	36.667	26.667	43.333	40.000	40.000	50.000	50.000	36.667
SA	Overall	41.111	35.778	31.111	50.000	47.778	44.444	53.333	54.222	38.889
ALL	High	40.222	44.444	54.444	46.667	50.000	62.222	45.778	54.222	54.444
ALL	Medium	58.889	48.889	49.111	50.000	53.333	52.222	51.111	53.333	54.444
ALL	Low	51.111	44.444	51.333	46.667	43.333	60.000	48.000	48.889	52.222
ALL	Overall	50.074	45.926	51.630	47.778	48.889	58.148	48.296	52.148	53.704

Table C4: Performance (%) of Gemma 4B fine-tuned on sparse-context training data across question types and evaluation setups (Zero-shot and Few-shot).

Ques. Type	Complexity	Zero Shot			Few-Shot (3 images)			Few-Shot (9 images)		
		None	Sparse	Detailed	None	Sparse	Detailed	None	Sparse	Detailed
LR	High	32.667	40.000	50.000	43.333	43.333	50.000	40.000	46.667	40.667
LR	Medium	53.333	40.667	36.000	36.667	43.333	43.333	46.667	46.667	50.000
LR	Low	43.333	36.667	47.333	37.333	33.333	50.000	43.333	33.333	40.000
LR	Overall	43.111	39.111	44.444	39.111	40.000	47.778	43.333	42.222	43.556
IE	High	53.333	56.667	76.667	40.000	53.333	86.667	50.667	60.000	90.000
IE	Medium	76.667	66.667	70.000	56.667	60.000	66.667	46.667	56.667	70.000
IE	Low	63.333	56.667	73.333	46.667	50.000	83.333	50.000	56.667	80.000
IE	Overall	64.444	60.000	73.333	47.778	54.444	78.889	49.111	57.778	80.000
SA	High	43.333	36.667	29.333	50.000	46.667	50.000	40.000	46.667	40.000
SA	Medium	46.667	33.333	33.333	46.667	56.000	46.667	53.333	53.333	43.333
SA	Low	33.333	46.667	36.667	43.333	40.000	40.000	43.333	50.000	38.667
SA	Overall	41.111	38.889	33.111	46.667	47.556	45.556	45.556	50.000	40.667
ALL	High	43.111	44.444	52.000	44.444	47.778	62.222	43.556	51.111	56.889
ALL	Medium	58.889	46.889	46.444	46.667	53.111	52.222	48.889	52.222	54.444
ALL	Low	46.667	46.667	52.444	42.444	41.111	57.778	45.556	46.667	52.889
ALL	Overall	49.556	46.000	50.296	44.519	47.333	57.407	46.000	50.000	54.741

Table C5: Performance (%) of Gemma 4B fine-tuned on detailed-context training data across question types and evaluation setups (Zero-shot and Few-shot).