POLITECNICO
MILANO 1863

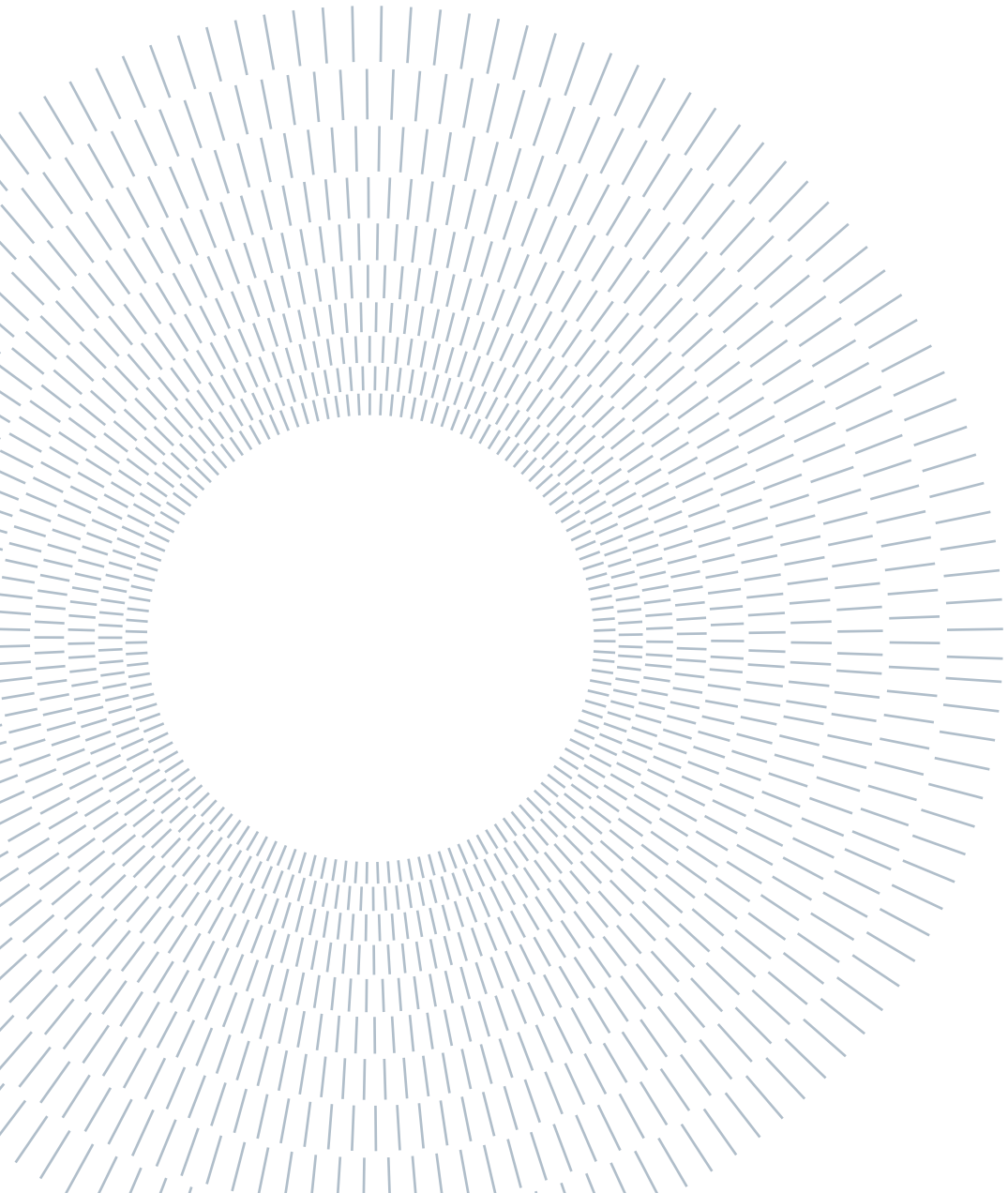
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

RAG-Flow: A Multimodal Retrieval-Augmented Generation Pipeline for Technical Manuals

Tesi di Laurea Magistrale in
Computer Science and Engineering
Ingegneria Informatica

Author: **Ziyan Cheng**

Student ID: 10789417
Advisor: Prof. Elisabetta Di Nitto
Academic Year: 2025-26

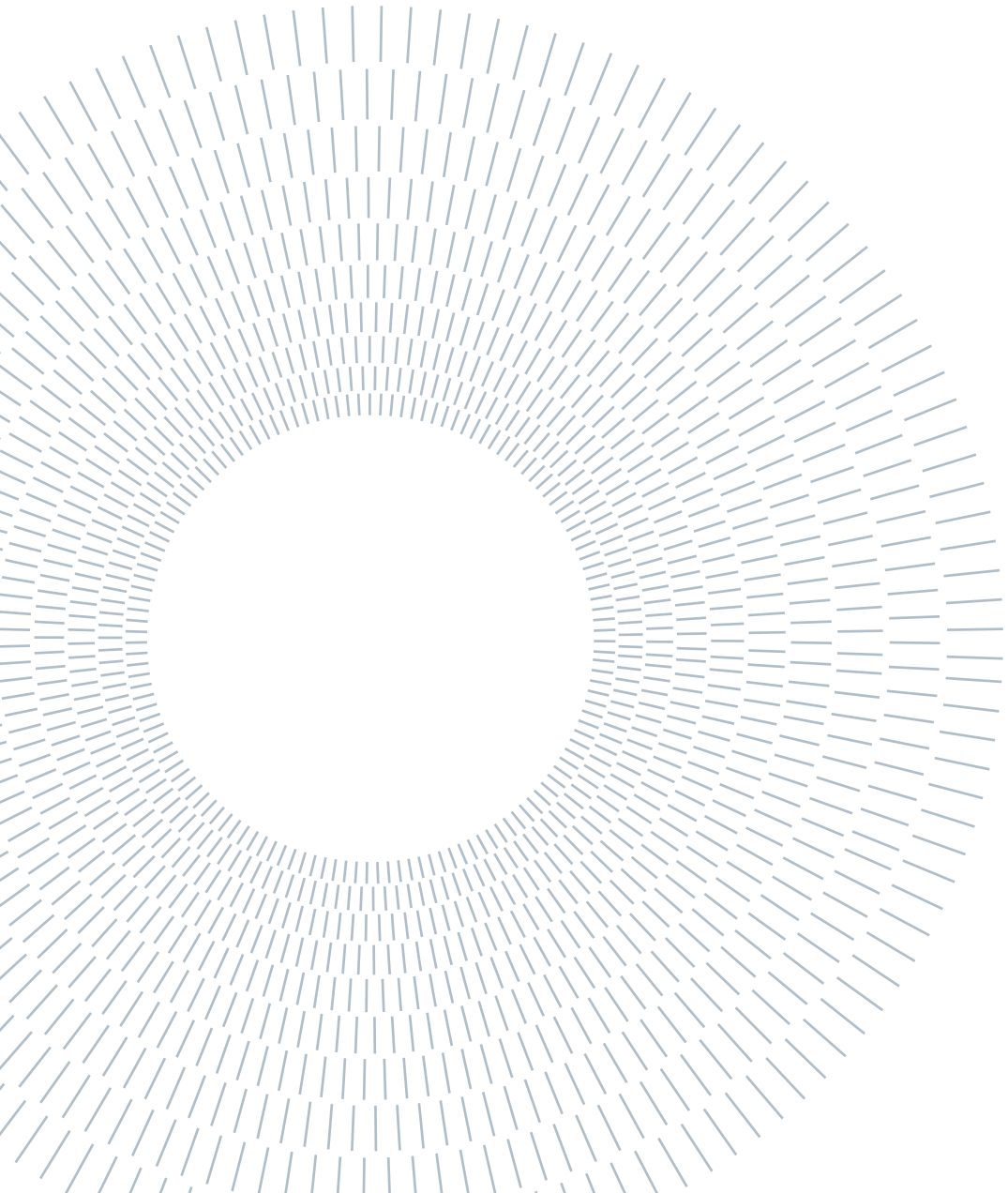


Abstract

Technical manuals make retrieval-augmented generation harder than ordinary passage retrieval. Useful evidence may appear as prose, table rows, screenshots, icons, drawing labels, page layout, or cross-page procedures. This thesis presents RAG-Flow, a multimodal pipeline that converts technical-manual PDFs into auditable evidence for retrieval and question answering. The evaluation was built in stages rather than as a single final run: v1 established the pipeline and visual-enhancement baselines; v2 tightened the artifact contract and tested chunking, retrieval, answering, and presets on 14 PDFs; v3 kept the v2 pipeline fixed and checked whether the remaining weakness could be explained by the text-retrieval backbone alone. The final v2 validation uses a 50-question search set, a 200-question validation set, and three separate Codex 5.5 extra-high review passes per answer.

The selected online contract is deliberately limited. It uses section-aware `1500/150/150` chunks, dense plus sparse text retrieval, and six runnable presets: `low`, `medium`, `high`, `low-with-image-input`, `medium-with-image-input`, and `medium-with-visual-recall`. On the 200-question validation set, `high` obtains the best mean score, 2.4133, while `low` reaches 2.3833 with much lower average retrieved context, 4156 tokens. Some negative results were as important as the winning row: the 24k context setting often crossed the serving limit, thinking-on sometimes produced empty answers, and image input could make visual-counting questions worse rather than better. Direct-PDF and v3 diagnostics suggest that the remaining gap is not only a text-backbone issue; it also comes from evidence preservation, evidence selection, and answerer capability.

Keywords: retrieval-augmented generation, multimodal RAG, technical manuals, PDF understanding, visual question answering, Qdrant, ColPali

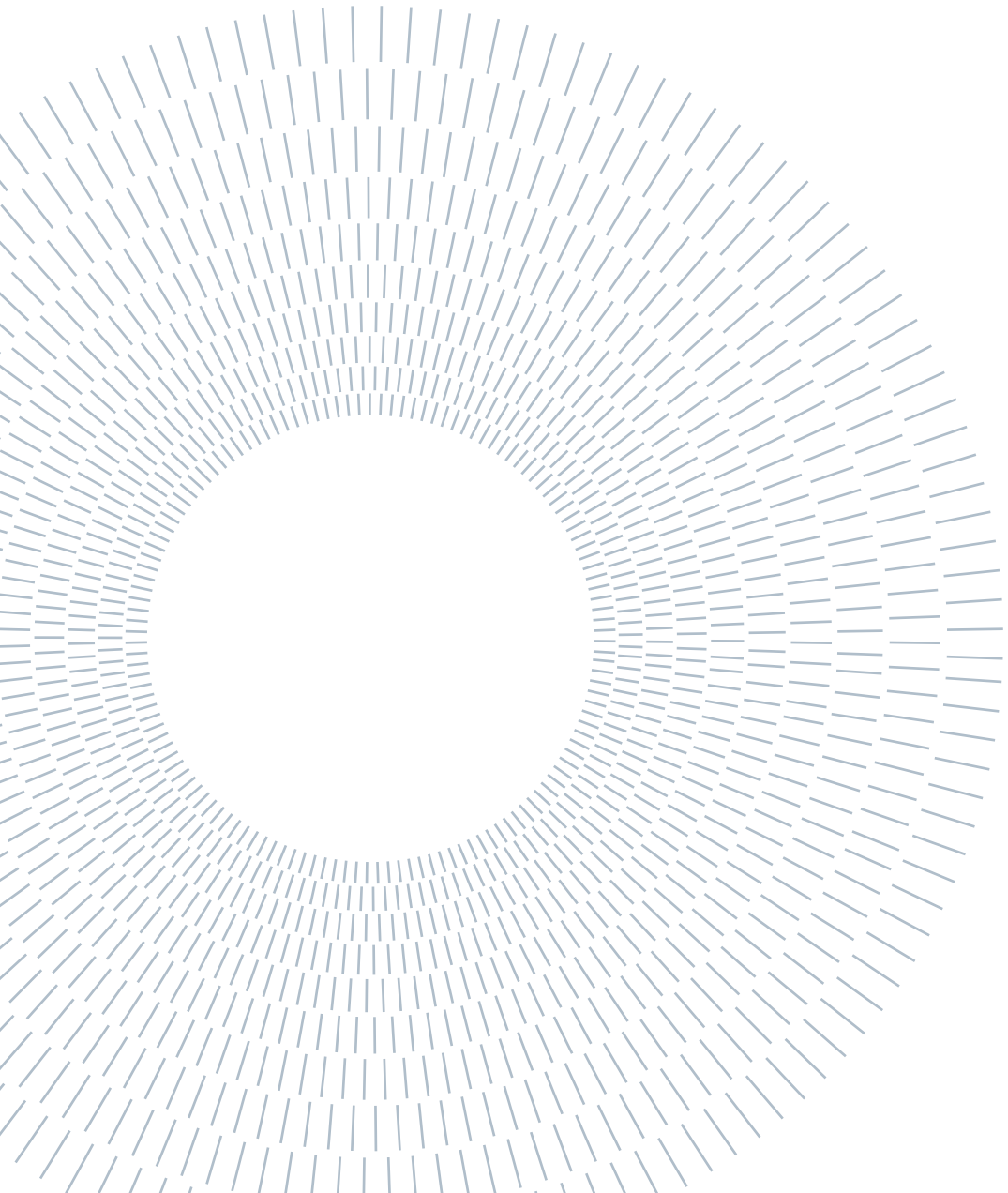


Abstract in lingua italiana

I manuali tecnici rendono la retrieval-augmented generation più difficile del normale retrieval di passaggi. L'evidenza utile può trovarsi in testo, righe di tabella, screenshot, icone, etichette di disegni, layout di pagina o procedure su più pagine. Questa tesi presenta RAG-Flow, una pipeline multimodale che trasforma PDF di manuali tecnici in evidenza verificabile per il retrieval e la generazione di risposte. La valutazione è stata costruita per fasi, non come un'unica esecuzione finale: v1 stabilisce la pipeline e le baseline di arricchimento visuale; v2 raffina il contratto degli artefatti e testa chunking, retrieval, answering e preset su 14 PDF; v3 mantiene fissa la pipeline v2 e controlla se la debolezza residua possa essere spiegata solo dal backbone di retrieval testuale. La validazione finale v2 usa un set di ricerca da 50 domande, un set di validazione da 200 domande e tre passaggi separati di revisione Codex 5.5 extra-high per ogni risposta.

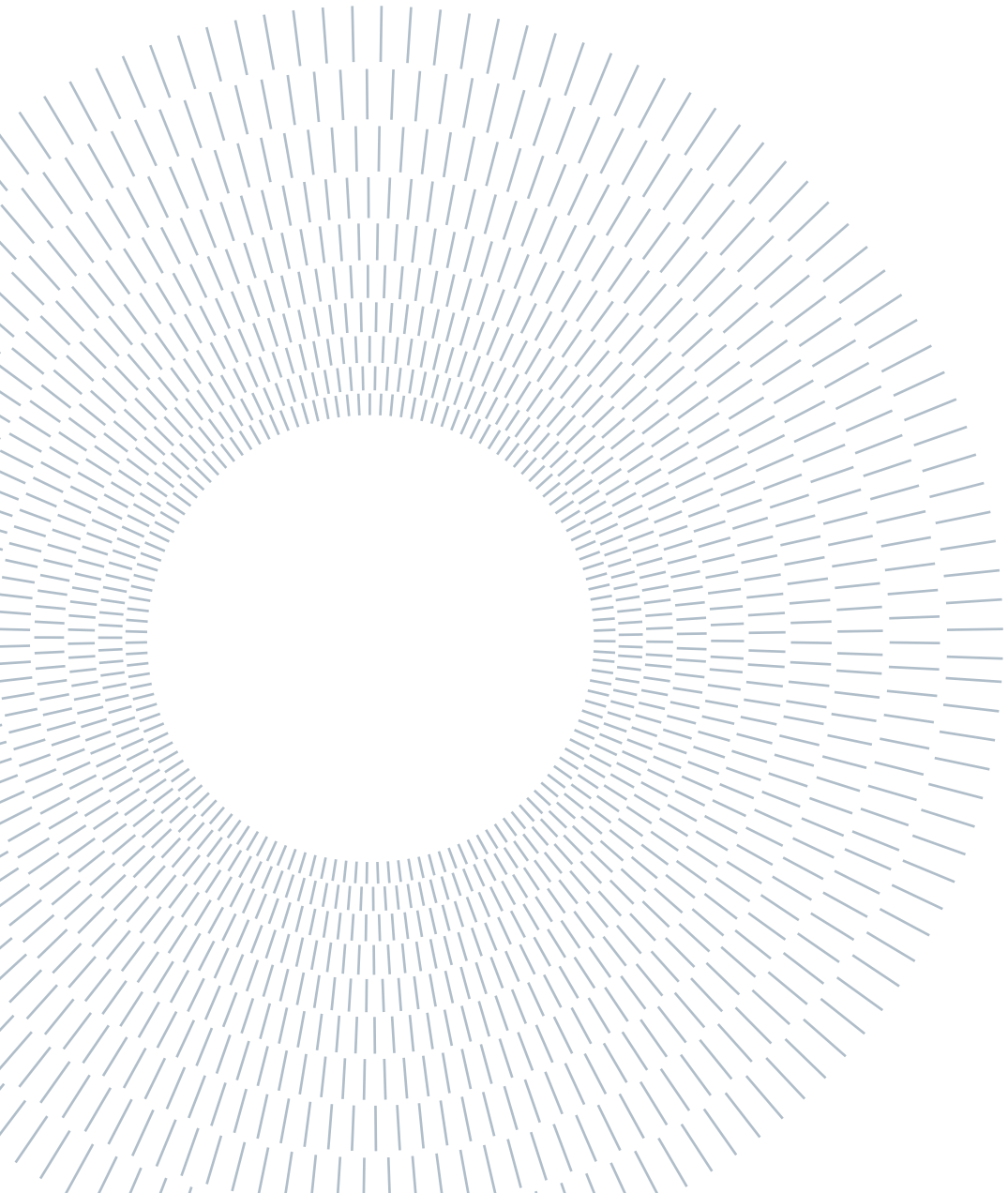
Il contratto online selezionato è volutamente limitato. Usa chunk sensibili alle sezioni `1500/150/150`, retrieval testuale denso e sparso, e sei preset eseguibili: `low`, `medium`, `high`, `low-with-image-input`, `medium-with-image-input` e `medium-with-visual-recall`. Nella validazione da 200 domande, `high` ottiene il miglior punteggio medio, 2.4133, mentre `low` raggiunge 2.3833 con un contesto recuperato medio molto più basso, 4156 token. Alcuni risultati negativi sono stati importanti quanto la riga migliore: il contesto da 24k spesso supera il limite di serving, il thinking-on produce talvolta risposte vuote, e l'input immagine può peggiorare domande di conteggio visuale invece di migliorarle. Le diagnostiche direct-PDF e v3 suggeriscono che il divario residuo non è solo un problema di backbone testuale, ma anche di preservazione dell'evidenza, selezione dell'evidenza e capacità del modello di risposta.

Parole chiave: retrieval-augmented generation, RAG multimodale, manuali tecnici, comprensione PDF, visual question answering, Qdrant, ColPali



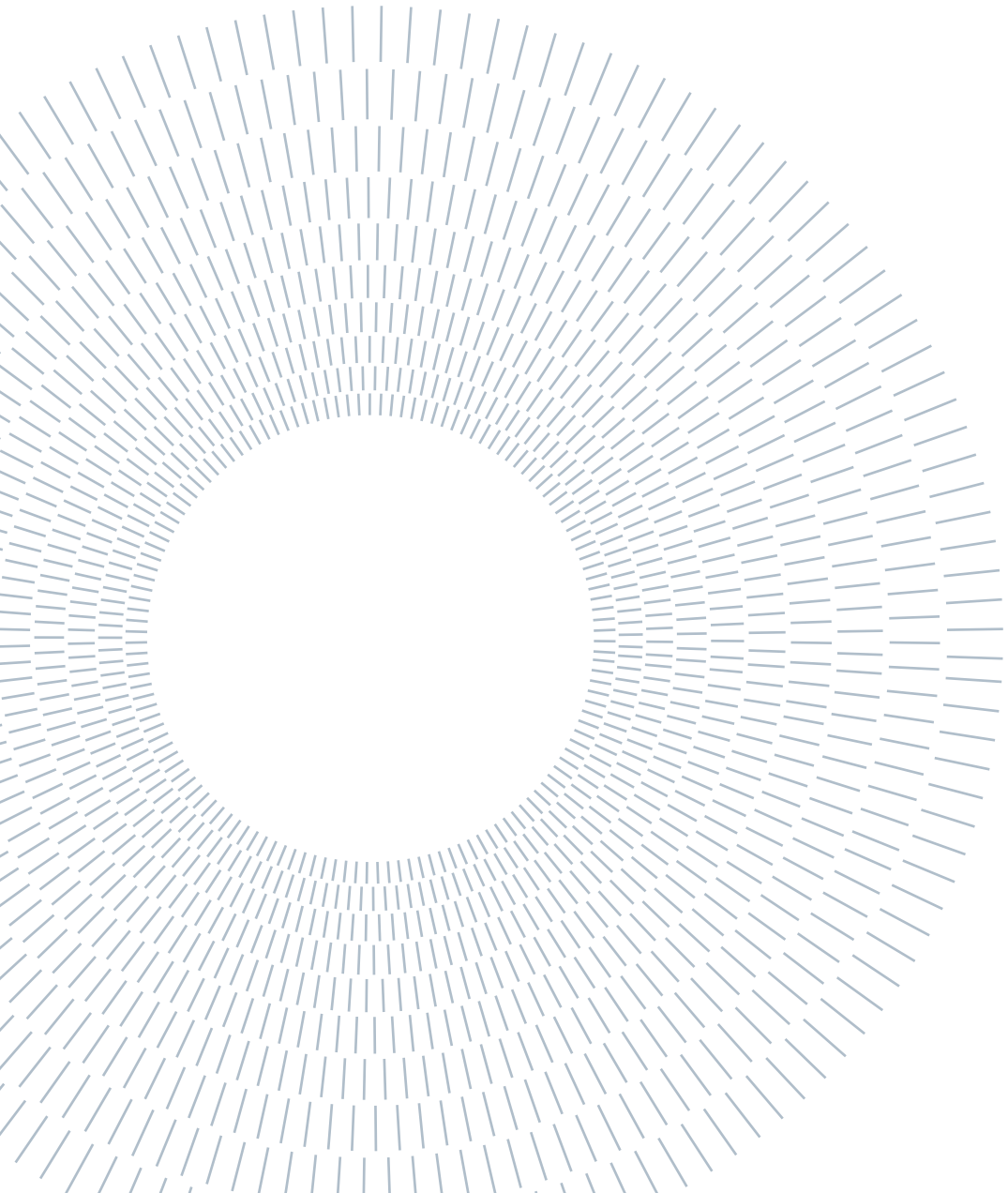
Declaration on the Use of Generative AI

Generative AI tools were used in this thesis in two distinct ways. First, they formed part of the experimental protocol for answer review and diagnostic comparison, as described in Chapters 5 and 9. Second, they were used to support language polishing, consistency checking, and revision of the manuscript. They were not used as a source of unverified technical claims or bibliographic references. The implementation, experimental design, interpretation of results, and final responsibility for the thesis remain the author's own.

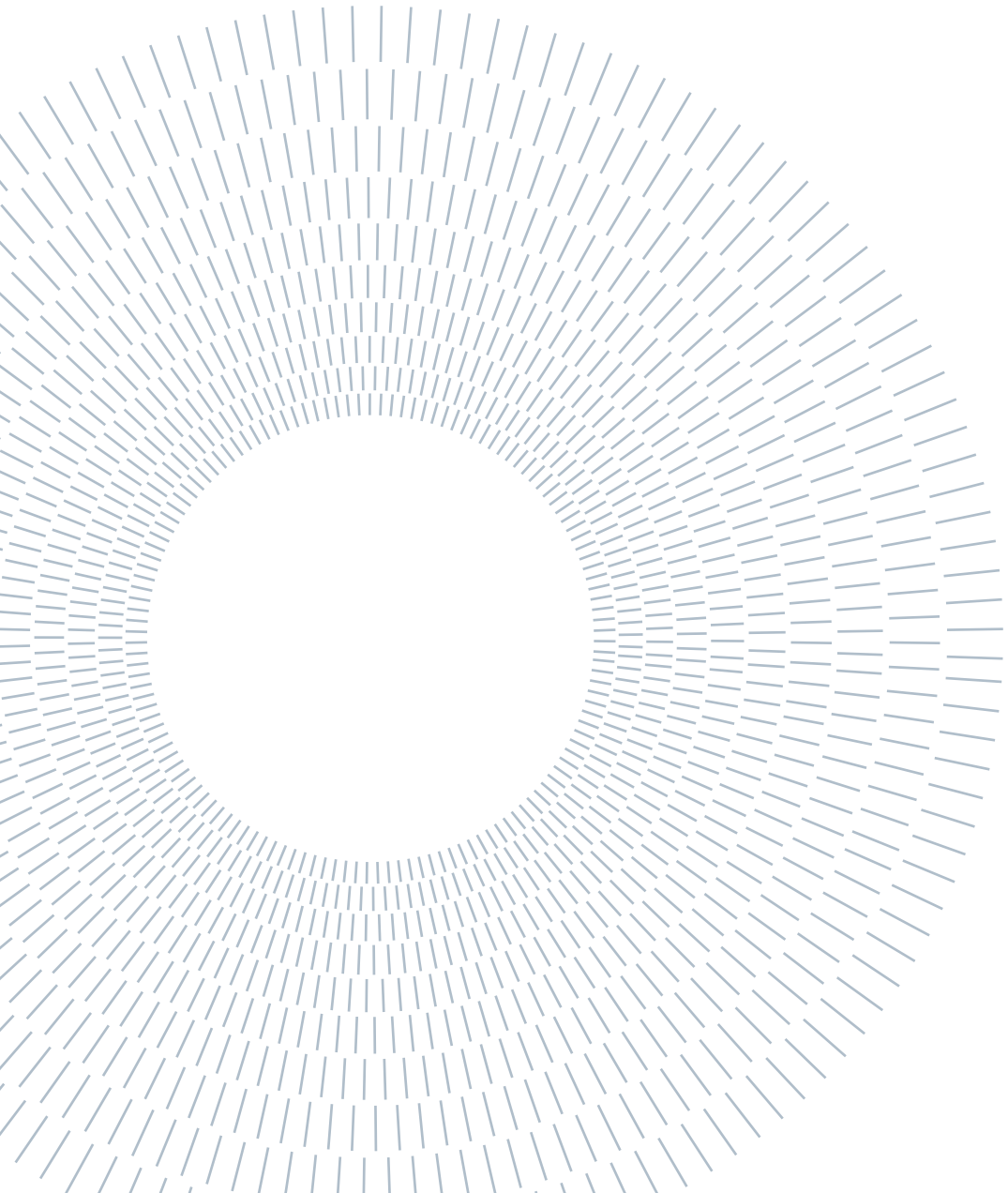


Contents

List of Figures



List of Tables



1 | Introduction

1.1. Motivation

Retrieval-augmented generation (RAG) is often introduced as a text problem: split documents into passages, embed them, retrieve the most relevant passages, and give those passages to a language model [6, 9]. Technical manuals do not fit that simplified setting. A camera, switch, or enterprise-software manual contains prose, tables, icons, screenshots, physical-dimension drawings, UI panels, and cross-page installation procedures. Many questions cannot be answered by isolated OCR text alone. For example, a switch dimension drawing may contain the correct values only as arrows and labels; a UI manual may refer to a small button icon that OCR omits; a quick-deployment guide may require joining a table row with a nearby figure caption.

RAG-Flow was built for this class of documents. Starting from source PDFs, it writes a sequence of artifacts: parsed content blocks, section metadata, patched small-icon text, image captions, section-aware chunks, Qdrant indexes, retrieved contexts, final answer payloads, and answer-quality reviews. The evaluation treats the system as a pipeline rather than a single model call, so parameter ownership, quality-sensitive settings, throughput-only settings, decision evidence, and operational constraints are recorded with the final command-line presets.

1.2. User-Level Goal and Core Terms

At the user level, the goal is simple: a support engineer, product specialist, or technical user should be able to ask task-oriented questions over a collection of manuals and receive an answer that is grounded in the source documents. The question may be about a product function, a configuration step, a compatibility condition, a table value, a diagram, or a visual installation detail. RAG-Flow is not designed to answer arbitrary open-web questions. It is designed to help users navigate a known technical-manual corpus while controlling how much evidence is sent to the language model.

The central resource is the **LLM context**: the retrieved text, captions, source breadcrumbs, page references, and optional image evidence that are placed in the prompt before the answer is generated. The **LLM output** is the generated answer. More context is not automatically better. Longer prompts cost more tokens, increase latency, can cross serving limits, and can make the answerer choose the wrong evidence among many similar manual passages. The system therefore optimizes the usage of the language model by selecting enough context to answer the question while avoiding unnecessary text and images.

Images have a specific role in this design. Technical manuals often encode important evidence in icons, screenshots, port diagrams, dimension drawings, and installation figures. RAG-Flow uses image understanding mostly during offline ingestion: small icons are repaired into text, and true image blocks receive captions and image-answering policies. This makes visual evidence searchable without sending every image to the model at query time. Online image input is kept as an explicit mode because it consumes additional tokens and, in the final experiments, does not improve average answer quality.

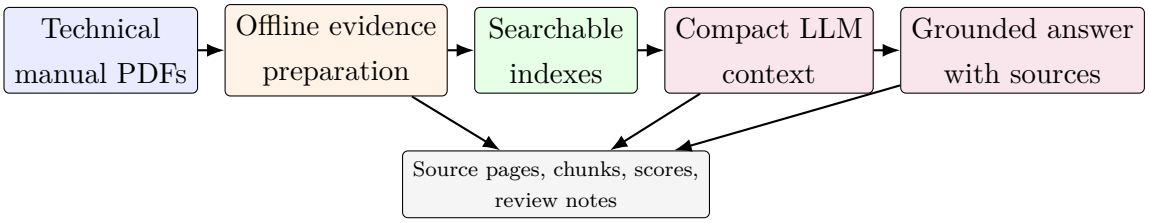


Figure 1.1: User-level view of RAG-Flow: manuals are prepared offline, retrieval selects a compact context for the LLM, and the generated answer remains auditable through source pages and saved artifacts.

Term	Meaning in this thesis
Technical manual	A product or software PDF containing procedures, specifications, diagrams, screenshots, tables, notices, or installation evidence.
RAG	Retrieval-augmented generation: retrieve source evidence first, then ask the language model to answer from that evidence.
LLM context	The evidence sent to the model before generation: retrieved text, captions, source breadcrumbs, page references, and optional image URLs.
LLM output	The final generated answer returned to the user and later scored by the review protocol.
Token usage	The approximate amount of input and output text processed by the model; it affects cost, latency, and serving-limit risk.
Chunk	A bounded evidence unit built from one or more document blocks. Chunk size and section boundaries determine what retrieval can pass to the answerer.
MinerU	The document-parsing tool used to convert PDFs into structured blocks, coordinates, images, tables, and page-level artifacts before RAG-Flow enrichment.
Qdrant	The vector database used to store dense, sparse, and optional visual retrieval points.
ColPali	A late-interaction visual retrieval model used to search rendered PDF pages when a question may depend on visual layout or diagrams.
Image input	Evidence images sent directly to the answering model at query time; useful as a diagnostic mode but not selected as the default.
Stage artifact contract	The required inputs, outputs, metadata, and rebuild rules that keep each stage auditable and prevent stale or mismatched evidence.

Table 1.1: Core terms used before the detailed system and experiment chapters

1.3. Why the First Version Still Matters

The project was developed through two core experimental generations and one later robustness check. The first version provides the original RAG-Flow baseline: MinerU parsing, PDF-outline section recovery, small-icon patching, image captioning, chunking and indexing, retrieval fusion, optional ColPali visual retrieval, and end-to-end answering. It also contains the early parameter studies for patching, captioning, retrieval coverage, and page-image answering. Those runs explain why several quality parameters are inherited instead of searched again in v2.

The second version does not replace the first; it tightens and extends it. It introduces more explicit stage artifact contracts, moves sectioning earlier, stabilizes source identity fields, uses a stricter Codex three-pass 0–5 scoring protocol, retests chunking/retrieval/answering on a multi-document source set, and performs a final 200-question validation of deployable presets. A third experiment set then tests whether the remaining weakness can be explained by the dense/sparse retrieval backbone alone. In the final thesis, v1 is used as the baseline evidence, v2 as the preset-validation layer, and v3 as a narrow robustness check on the retrieval-backbone interpretation.

1.4. Research Questions

The thesis answers six research questions:

1. How can technical-manual PDFs be converted into structured, section-aware, multimodal evidence while preserving source traceability?
2. Which v1 parameters are true quality baselines, and which are only throughput or serving adaptations?
3. Does v2 section-aware chunking improve final answer quality compared with the original larger v1 chunks and pure token windows?
4. After chunking and indexing are fixed, what retrieval parameters and text-retrieval backbone produce the best trade-off among answer quality, context tokens, latency, and deployment simplicity?
5. Should visual retrieval, retrieval-provided image input, or model thinking be enabled by default?
6. Which final presets should be provided as supported, runnable modes, and what are their expected token budgets and caveats?

1.5. Scope and Non-Goals

The work is limited to technical-manual question answering over PDFs. The source documents are vendor manuals, datasheets, installation sheets, diagrams, UI manuals, and related notices. The system is evaluated on this document type rather than on open-domain question answering, general visual question answering, or conversational chat. The evidence standard is source-grounded: a correct-looking answer is not enough if the answer cannot be supported by retrieved manual evidence.

Model training is outside the scope: the project does not train a new foundation model, embedding model, or visual retriever. It is also not an exhaustive full-factorial parameter search over the entire RAG design space. The contribution is the pipeline implementation, the stage artifact contracts, the reviewed multi-document evaluation protocol, and the preset choices measured under the serving environment used in the experiments. The claim is correspondingly bounded: for this PDF class, pipeline, and serving stack, the thesis identifies configurations with documented quality, cost, and caveats.

The experiments also separate offline and online cost. Offline visual enhancement, such as patching and captioning, can be expensive because it is paid once per document ingest and improves future retrieval. Online image input is different: it is paid for every question. The final design therefore keeps multimodal preprocessing, but does not turn image input on for every answer.

1.6. Contributions

The main contributions are:

- An end-to-end multimodal ingestion pipeline for technical manuals: parsing, sectioning, small-icon patching, image captioning, chunking, and indexing.
- A separation between quality parameters, stage artifact contracts, and throughput parameters. This separation prevents concurrency or batch-size adaptations from being misreported as algorithmic quality findings.
- A first-version experimental baseline covering patching quality, captioning throughput, chunk/retrieval coverage on 220 reviewed questions, and end-to-end answering with text and page-image inputs.
- A second-version methodology using a 50-question search set, a 200-question final set, and three separate Codex review passes per answer; the consolidated v2 archive contains 88 answer-bearing runs, 6001 generated answers, and 18003 review pass judgments.
- A third-version retrieval-backbone robustness check over dense-only, sparse-only, fused, RRF, final 200-question, and matched v2-high controls, showing that embedding replacement helps only modestly and does not by itself solve the high-recall evidence gap.
- A final parameter selection for chunking, indexing, retrieval, answering, and six runnable preset names, with validation status and caveats recorded for each:

`low`, `medium`, `high`, `low-with-image-input`, `medium-with-image-input`, and `medium-with-visual-recall`.

1.7. Final System at a Glance

The final conservative text baseline is the historical `default` preset, implemented with the engineering name `medium`. It uses section-aware chunks of 1500 tokens, overlap of 150 tokens, `k=80`, `top_k=20`, `rrf_k=10`, a 10k retrieved-context cap, no relative score pruning, `max_tokens=8000`, thinking disabled, and no image input to the answerer. It is neither the highest-scoring configuration nor the most token-efficient one; its role is to provide a stable baseline for interactive use.

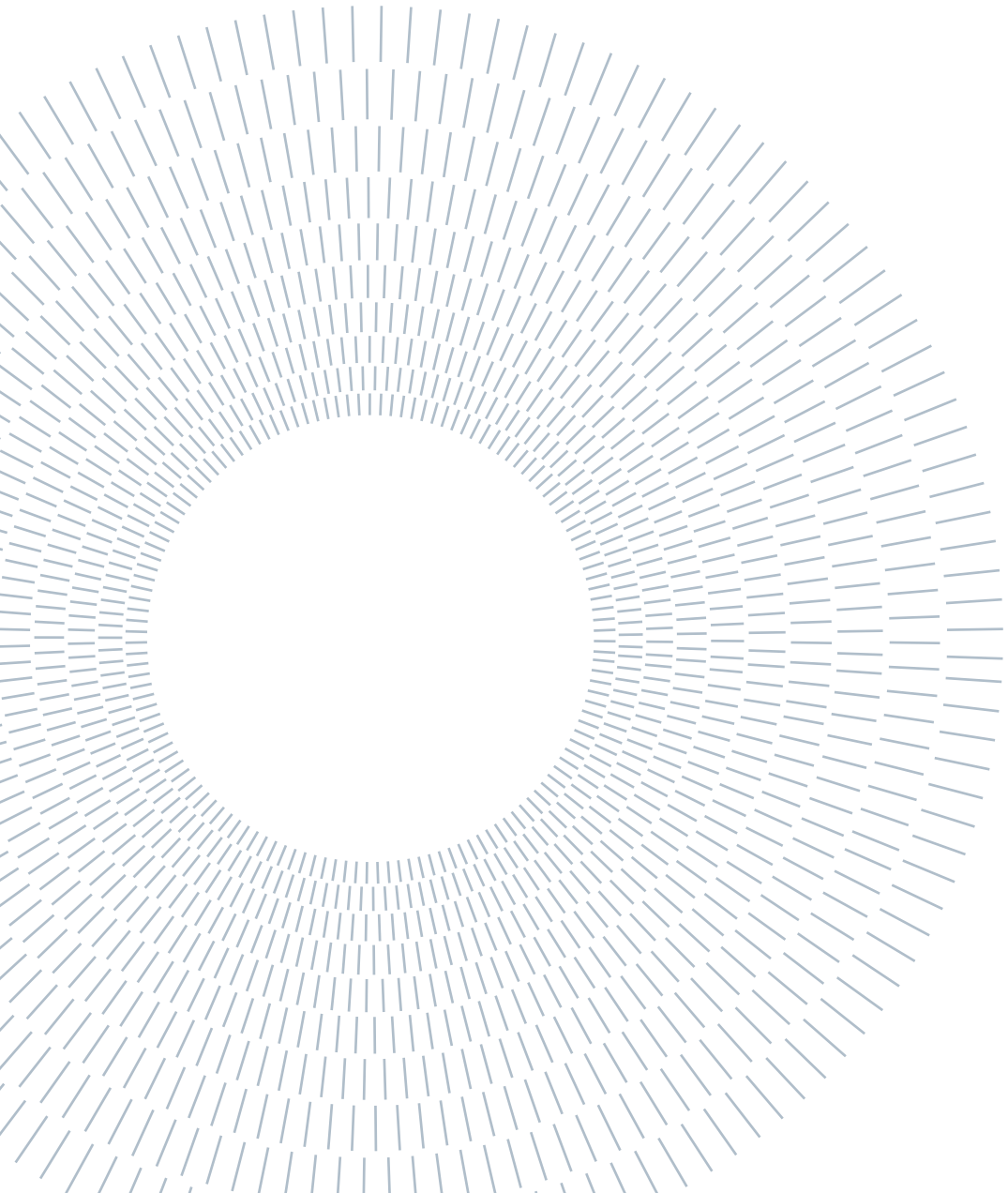
The best 200-question mean score is obtained by `high-recall`, whose engineering name is `high`: `k=150`, `top_k=80`, and a 16k context cap. Its mean score is 2.4133, with average retrieved context of 16584.94 tokens. The `compact` preset, whose engineering name is `low`, uses `k=150`, `top_k=10`, a 10k cap, and `score_ratio=0.4`; it reaches a 2.3833 mean score with only 4156.10 average context tokens. Among the validated presets, it is the most token-efficient option.

Several final choices come from negative results. A 24k context cap fails under the current 32768-token serving limit, producing 124 usage-missing cases out of 200 and a mean score of 0.8083. Thinking-on reduces quality and increases latency. Real image input reaches the model, but it does not improve the 200-question mean score and increases total tokens from 10588.75 to 14325.33 in the default comparison. Visual retrieval with naive page broadcast remains a recall mode, but its latency makes it unsuitable as the default.

1.8. Thesis Organization

Chapter 2 positions the thesis with respect to text RAG, dense and sparse retrieval, late-interaction visual retrieval, and multimodal answering. Chapter 3 describes the RAG-Flow pipeline, stage parameters, and artifact contracts. Chapter 4 explains the v1/v2 core research design, the v3 robustness-check role, sequential experiment strategy, experiment volume, and reproducibility model. Chapter 5 defines the datasets, review protocol, score scale, token-efficiency metrics, and failure taxonomy. Chapter 6 records the first-version experiments and explains which baselines carry into the final system. Chapter 7 gives the v2 chunking and indexing experiments. Chapter 8 reports the retrieval parameter search and the v3 text-backbone check. Chapter 9 reports the answering experiments, 200-question validation, and final presets. Chapter 10 discusses limitations and future

work, and the appendix contains extended tables and exact preset environment blocks.



2 | Background and Related Work

2.1. Retrieval-Augmented Generation

Retrieval-augmented generation combines a retriever with a generative model so that answers are grounded in external evidence rather than only in model parameters [9]. The standard formulation is usually text-centric: documents are split into passages, embedded, retrieved, and concatenated into a prompt. Dense passage retrieval made this pattern practical for open-domain question answering by replacing lexical-only matching with learned semantic vectors [6].

Technical manuals expose the limits of this simplified view. A single user question may depend on a table cell, a drawing label, an inline icon, and an adjacent step description. If the retrieval unit is too small, the answerer receives fragments; if it is too large, the model receives irrelevant neighboring procedures. In this project, retrieval settings are therefore judged through final answer quality rather than through hit-rate alone. Retrieval metrics remain diagnostic, but preset selection depends on generated answers and review scores.

A retriever can rank the correct page highly and still produce an unusable prompt if the chunk boundary cuts off the condition, table header, or figure caption that makes the evidence interpretable. Conversely, a configuration with slightly lower retrieval recall can be better in use when it gives the answerer shorter and more coherent evidence. For that reason, the experiments treat chunking, retrieval, and answering as a coupled pipeline rather than as independent benchmarks.

The thesis therefore avoids using a single “retrieval accuracy” number as the final objective. In classical text RAG, an experiment often stops once the relevant passage appears in the top ranked candidates. For a technical manual, this is not enough. A correct page may contain many similar UI procedures, repeated safety notes, or several tables with shared column names. If the final prompt contains the correct paragraph but also contains several visually similar paragraphs, the answerer can still choose the wrong operation or combine values from different rows. The evaluation asks a more operational question: after retrieval, can the deployed answerer produce a source-supported answer under a

realistic context and latency budget?

Several negative results are kept in the main argument because they decide where multimodality belongs in the pipeline. A result such as “image input increased tokens without improving mean score” is not just a failed setting; it separates one-time ingestion work from per-query answering cost. The system benefits from multimodal preprocessing because icons and captions are converted into searchable evidence once during ingestion. The same system can still reject always-on multimodal answering because that cost is paid at every user query.

2.2. Chunking as an Evidence Design Problem

Chunking is important because it decides what evidence the LLM can see. In practical terms, the retriever does not send an entire manual to the answerer. It sends a limited set of chunks. If the chunk is too small, the answerer may receive a value without the condition that explains it. If the chunk is too large, the answerer may receive many similar procedures and choose the wrong one. Chunking is therefore not only a preprocessing detail; in manual RAG it defines the evidence unit.

A good chunk must be large enough to preserve the local conditions that make a sentence meaningful, and small enough to avoid pulling unrelated procedures into the same retrieved block. A user question about adding a device, configuring an alarm, or reading a dimension drawing may require context from a heading, a table header, a figure caption, and one or two neighboring steps. At the same time, the next subsection may contain a procedure with similar menu labels but a different target action. This makes chunking both a semantic and an operational parameter.

RAG-Flow treats chunking as an experimental variable rather than as a fixed preprocessing constant. The first version used large 5000-token chunks because the main concern was not missing evidence in a single long manual. The second version retests that choice after sectioning becomes a formal upstream stage and after final answer quality becomes the main metric. The 1500-token section-aware profile performs better than the larger v1 profile in the completed runs, which matches the observed manual-QA failure mode: more context helps only when the answerer can locate and use the relevant part of it.

The problem is visible in PDFs with extracted tables and image descriptions. A table converted into text may contain repeated row labels; an image description may summarize a screenshot; a page header may repeat the same product name hundreds of times. A naive token window can place these items together without a meaningful document boundary.

Section-aware chunking uses the manual's outline as a weak semantic signal. It does not solve all layout problems, but it gives the retriever and answerer a better unit than a purely mechanical window.

2.3. Dense, Sparse, and Fused Retrieval

RAG-Flow uses dense and sparse text retrieval because technical manuals contain both paraphrasable instructions and exact terms. Dense embeddings help when the query paraphrases a manual section. Sparse retrieval remains important for device names, menu labels, part numbers, parameter keys, button names, and short codes. BM25-style sparse retrieval is a strong baseline for exact lexical evidence [13]. Reciprocal rank fusion is used because it gives a simple, robust way to combine ranked lists without training a task-specific reranker [2].

Late-interaction retrieval is also relevant. ColBERT showed that retaining token-level interaction can improve retrieval compared with single-vector document representations [7]. ColPali extends this idea to visually rich documents by embedding rendered pages and comparing query representations against page image patch representations [4]. This direction builds on the broader lesson from contrastive vision-language pretraining that visual and textual representations can support retrieval-style matching [12]. RAG-Flow adopts ColPali as an optional page-level visual route. In the final experiments, however, visual retrieval is kept as a recall mode rather than the default online path.

The dense/sparse split follows from the language of manuals, not only from a modeling preference. Some questions are naturally semantic, for example asking how to perform an operation whose wording differs from the manual's heading. Other questions are dominated by exact strings such as `S3006-4ET-60`, `RAG_FLOW_RETRIEVAL_K`, menu names, warning labels, or column headers. Dense retrieval can smooth over spelling and paraphrase, but it may underweight exact identifiers. Sparse retrieval can preserve those identifiers, but it does not understand paraphrase. Fusion keeps both signals without requiring a trained reranker for this specific corpus.

RRF is used because the system is intended to remain reproducible and explainable. The final presets use simple parameters such as retrieval `k`, final `top_k`, RRF constant, context cap, and score ratio. A learned reranker could potentially improve quality, but it would add another model, another training or calibration problem, and another source of hard-to-audit behavior. The thesis instead evaluates a hybrid retriever under section-aware chunking and answer-centered validation.

Retrieval signal	Concrete RAG-Flow implementation	Strength	Use or limitation
Dense text	FastEmbed `TextEmbedding`; model `intfloat/multilingual-e5-large`; 1024-d cosine Qdrant vector	Semantic matching and paraphrase tolerance	Always enabled for text retrieval; can dilute exact identifiers and short labels.
Sparse text	FastEmbed `SparseTextEmbedding`; model `Qdrant/bm25`; Qdrant IDF sparse modifier	Exact terms, part numbers, menu names, table labels	Always enabled and fused with dense retrieval; no custom BM25 `k1/b` tuning is used.
Visual page recall	ColPali model `vidore/colpali-v1.3-merged`; 128-d page multivectors with MaxSim	Visual recall for diagrams, screenshots, and visual sheets	Optional `visual-recall` variant; page-level signal is coarse and latency is higher.
Score-ratio pruning	`score_ratio=0.4` in `low`	Removes weak tail evidence after retrieval	Used only in the `low` preset; too aggressive pruning drops necessary context.

Table 2.1: Final retrieval signals and concrete implementations

The default path is kept text-only, and visual retrieval is exposed only as an explicit preset.

2.4. Document Understanding for Manuals

PDF manuals are not clean text corpora. Their semantics are distributed over layout, captions, drawing labels, screenshots, numbered figures, table structure, and repeated headers or footers. For retrieval, the pages first need a document-understanding layer. Prior document-AI work such as LayoutLMv3 and Donut shows that document understanding benefits from modeling text, layout, and visual content together rather than treating a page as plain text [5, 8]. In this thesis, MinerU is used to produce structured content blocks, page coordinates, image assets, and table/image fields [10]. RAG-Flow then adds stages that are specific to technical-manual RAG: outline-based sectioning, small-icon patching, image captioning, section-aware chunking, and evidence-oriented indexing.

The pipeline does not leave all visual understanding to the final answerer. Many visual signals are converted into text and metadata before indexing. Small icons are patched into text fields when OCR drops them. Image blocks receive descriptions and image-answering policies. Page images are indexed separately for optional visual recall. The default answering path can therefore remain text-only, while visual evidence is still available when the question requires it.

The gold set is anchored to original PDF pages and reviewed evidence notes rather than to extracted chunks, because extracted chunks are system artifacts that can change as the pipeline improves. This prevents the benchmark from rewarding a configuration merely because it matches a previous extraction format. It also makes visual and table questions

auditable: the reviewer can return to the source page when a generated answer appears plausible but lacks the required evidence.

The manual-specific preprocessing stages also reduce a common failure mode in multimodal RAG: pushing all document understanding to the final prompt. If the answerer receives only raw OCR text and several full-page images, it must simultaneously identify the relevant image, interpret the layout, read small labels, connect that visual evidence to a question, and formulate the answer. RAG-Flow distributes those responsibilities. Patching handles small inline symbols when the document is ingested. Captioning summarizes true image blocks and records whether an image is likely to be needed during answering. Sectioning gives every block a source path and breadcrumb. Chunking turns those enriched blocks into retrieval units. The final answerer then operates on evidence that has already been normalized and audited.

Visual reasoning is still needed in some cases. A dimension drawing with no extractable text, a port diagram, or a UI screenshot can require image evidence. The final system therefore keeps image-input and visual-recall variants, but exposes them as explicit modes instead of silently enabling them for every question.

2.5. Multimodal Answering and Operational Cost

Modern multimodal language models can consume image inputs, but the thesis results show that passing images is not automatically beneficial. In v1, rendered PDF pages increased tokens and latency without beating the best text-only configuration. In v2, retrieval-provided image input was repaired and verified: 1378 image URLs reached the answerer in the final 200-question image run. Even then, mean score decreased from 2.3383 to 2.3183 while average total tokens increased from 10588.75 to 14325.33.

The negative image-input result is about the tested operational form, not about images in general. Sending many full evidence images is less efficient than using patched text, captions, and text retrieval as the default. A more plausible future design is localized image evidence: crop only the region that corresponds to the retrieved chunk or visual hit, then send a small number of targeted images. That direction is consistent with the ColPali page-level result: visual page retrieval can help find candidate pages, but page-level visual scores are not yet precise enough to justify unfiltered image input for every answer.

Cost is part of the method, not only an engineering afterthought. If two systems obtain similar answer scores, the system with fewer tokens, lower latency, fewer moving parts,

and a simpler failure surface is the better default. The thesis therefore reports context tokens, total LLM tokens, image URL counts, latency, usage-missing cases, and failure categories. A multimodal system can score well in isolated examples while becoming less reliable under repeated service constraints. The final preset design reflects this operational constraint: `medium` is conservative; `high`, `low`, `medium-with-visual-recall`, and image-input variants are named tools for specific operational situations.

2.6. Evaluation and Auditability in RAG Systems

RAG evaluation is included because the industrial problem is not only to retrieve a passage, but to produce an answer that a user could safely act on. A generated answer can be wrong for several reasons. The retriever may fail to supply the required evidence; the answerer may ignore evidence that is present; a long context may contain the answer but bury it under noise; an unsupported answer may be factually plausible but not grounded in the retrieved context. Recent RAG evaluation work similarly separates answer correctness, faithfulness, and context quality rather than relying on one retrieval metric alone [3]. This thesis uses a review protocol that scores complete answers and records failure types. Retrieval metrics remain diagnostic, but they are not the final selection criterion.

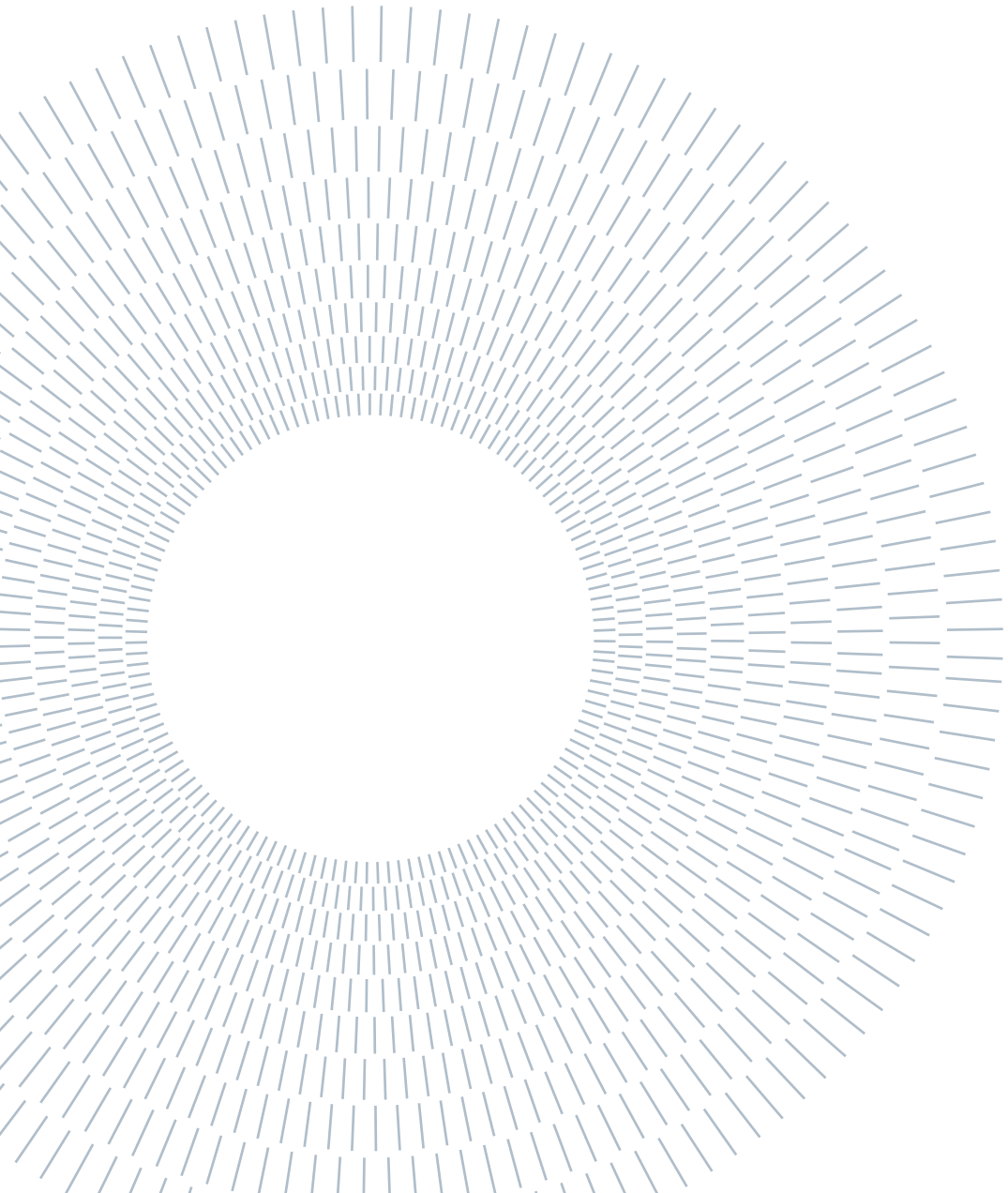
The three-pass Codex review protocol is designed to reduce single-review variance while preserving traceability. Each pass receives the question, canonical answer, system answer, a compact gold-evidence summary, and a clipped retrieved-context excerpt from the RAG run. The default scoring script clips that excerpt to 6000 characters. The final score is the mean of the three integer pass scores. Usage, latency, retrieved paths, and configuration are saved with the run and used for later analysis, but they are not direct reviewer prompt inputs. The protocol is less authoritative than a large blinded human evaluation, but it is stricter than self-grading and more informative than hit-rate alone. It also creates pass-level notes that can be inspected when a configuration is rejected.

Auditability also applies to artifacts. The final system keeps intermediate JSON files, Qdrant payloads, retrieval contexts, answer payloads, raw responses, usage records, and review notes. This artifact set is larger than a minimal prototype, but it is necessary for evaluating technical-manual QA. A system that cannot trace an answer back to a source page, a chunk id, and a retrieved context cannot convincingly claim source-grounded behavior.

2.7. Position of This Work

The thesis does not propose a new embedding model or foundation model. It studies a multimodal manual-RAG system through stage artifact contracts, auditable artifacts, reviewed QA protocol, v1 baseline, v2 parameter study, and final runnable presets. In this corpus and serving environment, answer quality depends on evidence preparation and conservative online defaults. Multimodal preprocessing helps; always-on multimodal answering is not justified by the measured results.

Compared with a conventional RAG prototype, RAG-Flow emphasizes details that determine whether a result can be trusted after the first demonstration: file naming contracts, stage boundaries, stale-index prevention, source identity, evidence pages, token budgets, serving limits, and repeatable run archives. The first user-facing demonstration is the interactive answering interface shown later in the system chapter, where the answer, research trace, retrieved sources, page references, scores, and direct PDF-opening controls are visible together. Compared with a pure document-understanding system, RAG-Flow evaluates whether extracted and enriched evidence improves downstream answers. Compared with a pure multimodal-answering experiment, it separates the value of multimodal preprocessing from the cost of sending images at query time. This positions the work as an applied systems study of technical-manual RAG.



3 System Design

3.1. Pipeline Overview

RAG-Flow is organized as an offline ingestion pipeline followed by online retrieval and answering. The offline stages convert PDFs into structured evidence and indexes. The online stages select evidence for a user query and generate an evidence-bound answer.

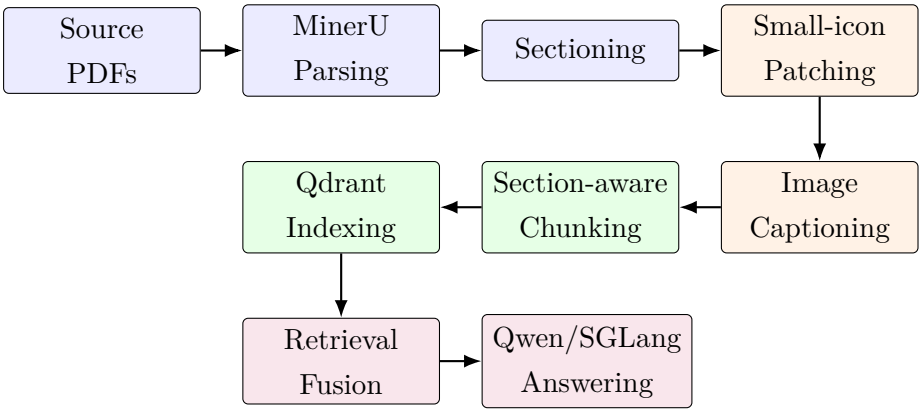


Figure 3.1: RAG-Flow offline build and online answering pipeline

The figure follows a source PDF through MinerU parsing, section recovery, small-icon patching, image captioning, section-aware chunking, Qdrant indexing, retrieval fusion, and Qwen/SGLang answering; colors separate offline preprocessing, indexing, and online answering.

The final stage sequence is:

```
source PDF
  -> MinerU parsing
  -> *_content_list.json
  -> sectioning
  -> *_content_list_SECTIONED.json
  -> patching
  -> *_content_list_SECTIONED_PATCHED.json
```

```
-> captioning
-> *_content_list_SECTIONED_PATCHED_CAPTIONED.json
-> chunking
-> *_content_list_SECTIONED_PATCHED_CAPTIONED_CHUNKED.json
-> indexing
-> retrieval
-> answering
```

The sequence is longer than a minimal RAG prototype because the experiments need to localize evidence loss. A shorter prototype could parse a PDF, split text, build an index, and answer questions, but it would hide which transformation caused a technical-manual failure. RAG-Flow makes each transformation visible. Sectioning is separated from parsing because source identity and outline breadcrumbs must be established before any LLM stage rewrites or enriches content. Patching is separated from captioning because inline icons and true image blocks have different semantics. Chunking is separated from indexing because chunk boundaries are a quality parameter, while index writing is a storage contract. Retrieval is separated from answering because the answerer must not silently choose additional evidence after the experiment has recorded a retrieval configuration.

The offline/online split also mirrors deployment cost. Parsing, patching, captioning, chunking, and indexing can be expensive because they are paid once per source document. Retrieval and answering are paid per user question. The thesis allows expensive offline visual enrichment while keeping the online default text-only, a choice supported by the measured v1 and v2 results.

3.2. Parameter Classes

The implementation and experiments distinguish three parameter classes.

Quality parameters change what evidence the model sees or how the evidence is represented. Examples include patching DPI, caption image side, chunk size, overlap, retrieval `k`, `top_k`, context cap, visual route, score ratio, output token cap, thinking, and image input.

Stage artifact contracts define required inputs, outputs, and metadata. Examples include the requirement that patching and captioning consume the current sectioned artifacts, and that chunks carry source identity, breadcrumbs, page indices, and optional image evidence.

Throughput parameters affect runtime, GPU pressure, or failure risk but should not be

interpreted as quality conclusions. Examples include concurrency, batch size, checkpoint frequency, server memory fraction, and timeout. They are recorded in logs, but they are not treated as algorithmic optima unless the experiment explicitly measures quality under that parameter.

Class	Examples	Affected object	Reporting mode
Quality parameters	DPI, image side, context tokens, chunk size, retrieval `k`, `top_k`, visual route, image input	Evidence content, evidence order, prompt length, and answer behavior	Main experimental tables, preset definitions, and rejection reasons.
Artifact-contract parameters	Required suffixes, source fields, breadcrumbs, chunk ids, page indices, bbox payloads	Whether stage artifacts remain joinable, auditable, and source-traceable	Stage artifact tables, rebuild rules, and provenance discussion.
Throughput parameters	Concurrency, batch size, checkpoint interval, timeout, memory fraction	Runtime, p95 latency, recovery cost, and GPU pressure	Runtime tables and serving notes; not promoted to quality claims unless quality is rechecked.

Table 3.1: Parameter classes used throughout the thesis

Keeping these classes separate prevents serving adaptations from being mistaken for evidence-quality conclusions.

The classification is used repeatedly in the experiments. For example, patching DPI is a quality parameter because it changes the rendered crop and the model's ability to recognize small icons. Patching concurrency is a throughput parameter because it changes how many requests are in flight but not which crop is shown to the model. Chunk size is a quality parameter because it changes the prompt evidence. Qdrant collection names and chunk ids are artifact-contract parameters because mixing old and new payloads can break provenance even if the answer text appears plausible.

3.3. Stage Parameters and Artifact Contracts

The word *contract* is used in a narrow engineering sense. For each stage, RAG-Flow records which artifact it consumes, which artifact it writes, which quality parameters can change the evidence, and which metadata must be preserved for audit. This is easier to inspect than a single monolithic pipeline setting: if an answer fails, the run archive can show whether the loss happened during parsing, patching, captioning, chunking, retrieval, or answering.

Stage	Main quality parameters	Main output	Artifact and audit rule enforced in v2
Parsing	Parser choice and PDF rendering assumptions	Raw content list, origin PDF, extracted image assets	MinerU converts each PDF into structured blocks, coordinates, images, and tables. This stage does not repair OCR, infer sections, or caption images.
Sectioning	Outline recovery and source identity policy	Sectioned content list and sectioning audit	Writes source identity and outline-derived breadcrumbs before any LLM stage rewrites or enriches content.
Patching	Render DPI, icon-repair policy, fallback policy	Patched sectioned content list and patching view PDF	Accepts only current sectioned artifacts; preserves original text when icon repair is unsafe.
Captioning	Maximum image side, context tokens, image-answering policy	Captioned patched content list and captioning view PDF	Captions true image blocks and records whether an image may need to be sent at answer time.
Chunking	Maximum chunk tokens, overlap tokens, minimum chunk size, section awareness	Chunked content list	Uses existing source fields and section breadcrumbs; chunk text changes require text reindexing.
Indexing	Dense model, sparse model, visual index mode, batch settings	Qdrant text and optional visual points	Text points are chunk-level; visual points are page-level; payloads must match the current chunk ids.
Retrieval	Retrieval `k`, final `top_k`, RRF constant, context cap, score ratio, visual route	Context text, selected chunks, optional image URLs	Selects a bounded LLM context under explicit evidence-selection rules saved with the run.
Answering	Answer model, output token cap, thinking flag, image input flag	Qwen answer, usage, latency, answer artifact	Does not re-query Qdrant or inspect PDFs; it consumes only retrieval-provided evidence.

Table 3.2: Stage-level parameters and artifact contracts

File naming and metadata continuity are treated as part of the system contribution, because they determine whether later answers can be traced back to source pages. The table also explains why the thesis separates stage parameters from throughput parameters. Retrieval `k` changes the evidence available to the LLM; patching concurrency only changes how quickly artifacts are produced. Only the former can justify a quality conclusion.

3.4. Source Identity and Provenance Model

The v2 pipeline introduces a stable source identity model because the final corpus contains multiple long manuals and short visual sheets. Absolute file paths cannot be used as semantic identifiers: they change across machines, remote servers, and local Google Drive mounts. Instead, every relevant block and chunk receives a normalized `source_relpath`, a `source_filename`, page indices, and a human-readable breadcrumb. These fields are written before patching and captioning so that later LLM-generated content remains attached to the correct source.

The provenance model has three layers:

- **Document identity**: normalized source family and PDF filename, independent of local absolute path.
- **Structural identity**: outline-derived breadcrumb, section path, section level, page indices, and block indices.
- **Retrieval identity**: chunk id, Qdrant point id, vector names, score fields, selected rank, and optional visual-page references.

Field family	Examples	Written by	Used by
Source fields	`source_relpath`, `source_filename`	Sectioning	QA mapping, retrieval display, review audit, cross-PDF comparisons.
Section fields	`breadcrumb`, `section_path`, section title and level	Sectioning	Chunk grouping, context labels, answer citations, failure analysis.
Page/block fields	`page_indices`, `block_indices`, bbox payloads	Parsing and chunking	Source-page audit, visual routing, image evidence selection.
Chunk fields	`chunk_id`, `chunk_idx`, `chunk_mode`	Chunking	Qdrant text points, retrieval output, run comparison.
Image evidence fields	`image_answering_policy`, image path, confidence, reason	Captioning and chunking	Optional image-input presets and visual failure analysis.

Table 3.3: Provenance fields carried across the final pipeline

These fields make multi-document retrieval auditable rather than only searchable.

The source identity model also explains why v2 rejects silent fallback behavior. If a stage runs on an old input file that lacks breadcrumbs, it may still produce text that looks reasonable, but the final answer cannot be reliably traced. In a manual-QA thesis, this is a serious error: the answer must be judged not only by fluency but also by source support.

3.4.1. Parsing

Parsing uses MinerU to create a structured content list, layout artifacts, extracted images, and an origin PDF copy. This study does not retune MinerU parsing. The relevant output is `*_content_list.json`, which contains block text, page indices, bounding boxes, image paths, and table/image fields. MinerU is treated as the document-understanding backend feeding the RAG-specific stages [10].

The parser output is deliberately considered an intermediate artifact, not ground truth. It may contain OCR omissions, table linearization artifacts, repeated headers, or image blocks whose surrounding text is insufficient. The later stages are designed to enrich and audit these outputs, not to assume they are already ready for retrieval. Accordingly, the evaluation gold set is built from original PDFs and rendered pages rather than from MinerU content lists.

3.4.2. Sectioning

Sectioning is promoted in v2 from an explanatory helper to a formal pipeline stage. It reads the PDF outline/bookmarks with PyMuPDF and writes stable source and section metadata into every block. It does not call an LLM to guess missing headings. If a PDF has no outline, sectioning still writes source fields and lets chunking fall back to token-window mode.

Each block receives at least:

```
source_relpath
source_filename
breadcrumb
```

If outline matching succeeds, it also receives section path, title, level, source, confidence fields, and the matched outline index. This makes evidence audit independent of local absolute paths while avoiding reliance on local machine-specific file names.

When a PDF has no useful outline, sectioning still writes source identity fields. This is needed for visual one-page sheets such as dimension drawings and port diagrams, where the document may have no table of contents but still needs stable provenance. In those cases, chunking falls back to token-window behavior while preserving source and page identity.

3.4.3. Patching

Patching repairs small inline icons that OCR often omits from text fields. It is not a general OCR rewrite step. Its input and output artifacts are:

```
*_content_list_SECTIONED.json
*_content_list_SECTIONED_PATCHED.json
```

Its v1 quality baseline is inherited:

```
dpi = 250
page_window_size = 200
max_new_tokens = 8000
```

The v2 change is contractual: missing sectioned input is an error, no-missing output must contain the strict token `NO_MISSING_SAFE`, empty icon markers are rejected, and only-icon fallback inserts valid markers back into the original text rather than deleting the original text.

Patching checks only fields where inline icon loss is likely to affect meaning: text fields, list items, table bodies, table footnotes, and image footnotes. It avoids headers, footers, page numbers, and most title fields because those often repeat across pages and can inject noise. The crop is built from the target field bbox, and when MinerU has detected nearby inline icon candidates the patcher uses the union of the target and icon boxes. This keeps the visual evidence local while allowing the VLM to see the missing symbol.

The patching prompt is intentionally conservative. It asks for missing icons to be inserted as `[Icon: name]` markers while preserving the original text. If the model returns explanatory prose, an empty marker, or a response that drops the original text, the output is rejected or repaired by fallback. The final system prefers preserving imperfect OCR text over accepting a fluent but contaminated patch, because retrieval can tolerate some missing icons more easily than it can tolerate LLM explanations injected into the document body.

3.4.4. Captioning

Captioning only processes true image blocks, not inline icons. Its input and output artifacts are:

```
*_content_list_SECTIONED_PATCHED.json
*_content_list_SECTIONED_PATCHED_CAPTIONED.json
```

The inherited v1 quality baseline is:

```
max_image_side = 2048
max_context_tokens = 2000
max_new_tokens = 8000
```

The v2 change is that contextual text is section-aware. If an image belongs to a section, neighboring text is collected only from the same section. The breadcrumb is passed as document context without consuming the before/after context budget. The model writes `image_description_vlm` and an `image_answering_policy` among `caption_only`, `image_optional`, `image_recommended`, and `image_required`.

Captioning has two outputs with different purposes. The first is the plain-language description that enters text retrieval. The second is the image-answering policy that tells retrieval whether the original image may be needed by a multimodal answerer. Separating these two avoids an all-or-nothing design. A screenshot can be well described in text and need no image input; a dense wiring diagram may receive a caption but still be marked image-recommended because exact visual inspection could matter. The final experiments show that this metadata supports diagnostics even though image input is not selected as the default.

3.4.5. Chunking

Chunking converts sectioned, patched, and captioned blocks into retrieval units. In `auto` mode, section metadata is used when available; otherwise the system falls back to token windows. Each chunk carries source identity, section path, page indices, block indices, and optional image-answering evidence.

The final chunk payload is intentionally redundant. It stores human-readable fields such as `breadcrumb` and `section_title`, machine-joinable fields such as `chunk_id` and `block_indices`, and page-local fields such as `page_indices` and `bboxes_by_page`. The redundancy is deliberate: the retriever needs vector ids and score fields; audit scripts need page and evidence provenance; the interface needs readable breadcrumbs; the visual route needs page and bbox links.

```
{
  "chunk_id": "...",
  "source_relpath": "DSS/manual.pdf",
  "source_filename": "manual.pdf",
  "breadcrumb": "DSS > manual.pdf > 3 Basic Configurations",
  "section_path": ["3 Basic Configurations", "3.1 Managing Resources"],
```

```
  "page_indices": [40, 41],
  "block_indices": [128, 129, 130],
  "image_answering_evidence": [...]
}
```

The chunker does not cross section boundaries merely to fill a token budget. If the current section ends, a smaller final chunk is acceptable. If a section continues, the chunker flushes only when adding the next block would exceed the maximum and the current chunk has passed the soft minimum. Overlap is taken from the tail of the previous chunk under the overlap-token budget. This prevents a large table or image description from being duplicated wholesale into the next chunk simply because it is the final block before a boundary.

3.4.6. Indexing

Indexing writes chunk-level dense and sparse text vectors and optional page-level ColPali multivectors into Qdrant [11]. Text indexing uses one point per chunk and two vector names:

```
chunk-text-dense
chunk-text-sparse
```

The dense text vector is generated with FastEmbed's `TextEmbedding` wrapper. The configured dense model is:

```
intfloat/multilingual-e5-large
```

During indexing, the exact `chunk_content` string is passed to `embed`; during retrieval, the user question is passed to `query_embed`. The Qdrant collection stores this named vector with 1024 dimensions and cosine distance. The sparse vector is generated with FastEmbed's `SparseTextEmbedding` wrapper. The configured sparse model is:

```
Qdrant/bm25
```

The resulting sparse indices and values are stored under `chunk-text-sparse`. In the repository, BM25 is not retuned with custom `k1` or `b` values. The explicit Qdrant-side sparse parameter is the IDF modifier, so the sparse vector schema is:

```
SparseVectorParams(modifier=IDF)
```

Visual indexing is page-level and uses:

```
page-image-colpali
```

The visual vector is not a section embedding. Its payload links visual page hits back to current chunk ids, page indices, source metadata, and section breadcrumbs.

Indexing is a consistency boundary. Text vectors represent the exact chunk text that was current at index time. Visual payloads reference the chunk ids and page payloads that were current at index time. If a chunking experiment changes text, ids, page references, or image evidence, the corresponding index must be rebuilt. Otherwise retrieval can return stale chunk ids while the final answer and review tables appear to use the new configuration.

3.4.7. Retrieval and Answering

Retrieval fuses dense, sparse, and optionally visual evidence. Dense retrieval captures semantic similarity; sparse BM25 captures exact device names, menu labels, and parameter strings [13]; reciprocal rank fusion combines ranked lists [2]; ColPali supplies page-level visual retrieval when enabled [4]. For text retrieval, dense and sparse searches use the same `retrieval_k`. RRF gives dense and sparse routes equal weight and scores each hit as `1/(rrf_k + rank + 1)` before final top-k, context-budget, and score-ratio pruning. Answering uses a local Qwen-compatible SGLang service [14] and receives only the retrieval final output. If image input is enabled, retrieval appends image URLs for evidence images marked recommended or required by captioning.

3.5. Final Inherited Baselines

Stage	Inherited v1 baseline	Final use
Patching	`dpi=250`; `batch_size=512`; `checkpoint=1`; default concurrency 8	Quality baseline retained; v2 tightens input and fallback contract.
Captioning	`max_image_side=2048`; `max_context_tokens=2000`; `batch_size=32`; v1 concurrency 6	Quality baseline retained; v2 changes context source to section-aware.
Chunking	`auto`; `max=5000`; `overlap=500`; `min=200`	Treated as the v1 high-recall baseline and retested against smaller v2 chunks.
Indexing	`text_batch=256`; `visual_dpi=200`; `visual_batch=8`	Fixed Qdrant baseline; not retuned in v2.
Retrieval	text-only; `k=150`; `top_k=80`; `rrf_k=10`; 10k cap	Starting point for v2 retrieval search.
Answering	no page images; thinking off	Baseline for retrieval experiments; then retested in v2 answering.

Table 3.4: v1 baselines carried into the final system

Quality baselines and runtime adaptations are reported separately.

3.6. Implementation Surface

RAG-Flow is operated through a command-line interface rather than through isolated research notebooks. This matters for reproducibility: the same entry points used for experiments are also the entry points used for reruns and deployment checks. Typical operations are:

```
rag-flow mineru run
rag-flow ingest --from-stage sectioning --to-stage chunking --force
rag-flow index text
rag-flow retriever
rag-flow chat
```

The implementation keeps every major artifact on disk. A run can be audited from the source PDF to the final answer because each stage writes an explicit intermediate file. For example, a final answer can be traced back through the retrieval output, Qdrant chunk ids, the chunked JSON, the captioned and patched content list, the sectioned content list, and the original PDF page. This artifact chain is detailed so that a plausible answer can still be traced to the source text, page, or image block that produced it.

The same audit contract is exposed in the interactive answering interface. Figure 3.2 shows a representative run in which the generated answer is followed by the research trace and the retrieved document sources. The interface keeps the source PDF names, page references, retrieval scores, and direct PDF-opening controls visible to the user, so the answer is not treated as an isolated chat message.

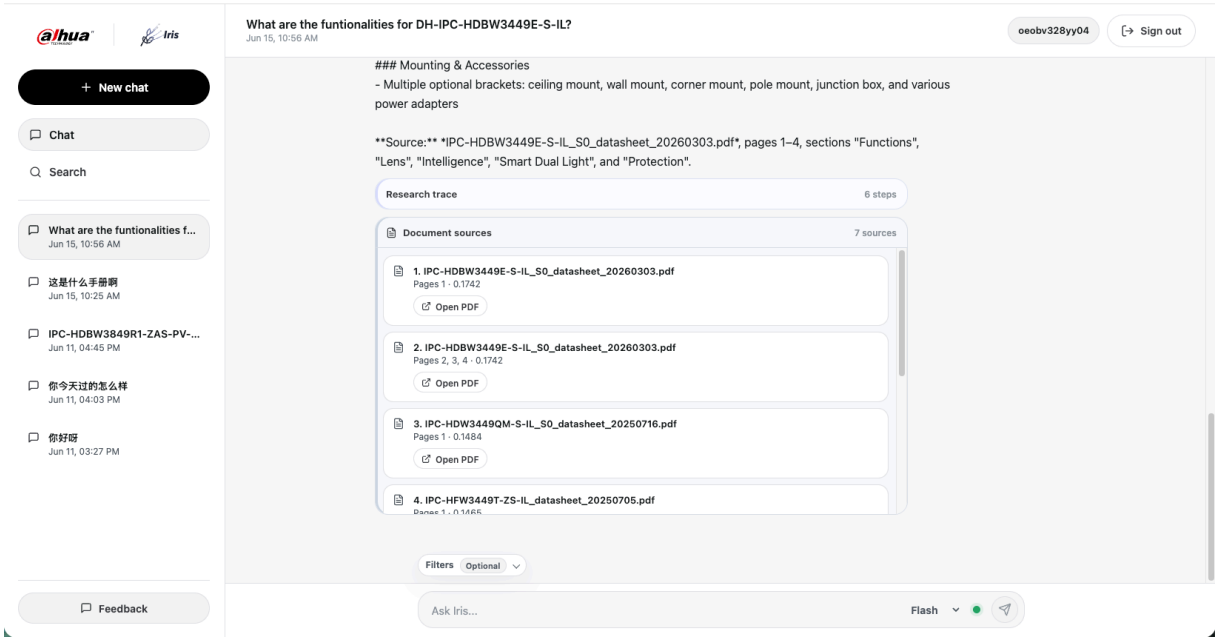


Figure 3.2: RAG-Flow answering interface with generated answer, research trace, retrieved document sources, page-level provenance, retrieval scores, and direct PDF-opening controls for manual audit.

The repository also separates local runtime configuration from quality parameters. Environment files and scripts record server paths, model paths, Qdrant locations, and SGLang endpoints. The thesis tables report the quality-relevant parameters: chunk size, retrieval candidate counts, context caps, visual route, score pruning, thinking, output budget, and image input. This distinction keeps deployment adaptations, such as GPU memory fraction or concurrency, from being mistaken for scientific conclusions.

3.7. Artifact Quality Checks

One artifact-level audit over the processed v2 corpus counted 14 PDFs, 9465 text blocks, 526 table blocks, 1406 image blocks, and 752 inline images. It also detected patching contamination cases where an LLM explanation leaked into patched text. The audit found 17 contaminated patched blocks and 15 contaminated chunks, concentrated mostly in the long DSS manuals. These artifacts are not used as a separate score for the final presets, but they help explain failure modes and motivate stricter v2 contracts.

Audit item	Count	Interpretation
Source PDFs	14	Same corpus as the final v2 evaluation.
Text blocks	9465	Main body evidence after parsing and enrichment.
Table blocks	526	Table extraction is substantial enough to require explicit table questions.
Image blocks	1406	Includes true images and visual artifacts; captioning filters true image blocks.
Inline images	752	Main motivation for small-icon patching.
Contaminated patched blocks	17	Cases where VLM explanation text leaked into patch output.
Contaminated chunks	15	Downstream chunks affected by those patching issues.

Table 3.5: Artifact-quality audit used to tighten the final stage contracts

The audit supports conservative patching fallbacks and source-aware review.

The contamination examples show why the patching contract has to be strict. Some VLM outputs explained the task before returning the patched text; others replaced a small OCR artifact such as a letter with an icon marker while dropping the original text. These are the kinds of errors that a naive LLM enrichment stage can hide. The v2 contract responds by requiring strict no-missing tokens, rejecting malformed markers, preserving original text under only-icon fallback, and treating patching as a quality-sensitive stage rather than a harmless formatting step.

3.8. Why the Contract Matters

The contract is strict because v1 showed that silent compatibility can break provenance. If patching accepts an old raw `content_list.json`, it can repair icons without the final

source identity fields. If captioning accepts an old patched file, image policies may not line up with the current section breadcrumbs. If indexing writes visual payloads from one chunked JSON and text payloads from another, ColPali page hits can point to stale chunk ids. These errors do not always crash the system. They can produce plausible answers with broken provenance, making the failure hard to audit.

For that reason, v2 makes missing upstream artifacts fail early. A file suffix is not treated as cosmetic: `SECTIONED`, `PATCHED`, `CAPTIONED`, and `CHUNKED` describe which semantic contract has been satisfied. This is also why the final experiments rebuild the relevant index whenever chunk text, chunk ids, page indices, bbox payloads, or image evidence change. Retrieval and answering parameters can vary without reindexing; chunking and evidence-payload changes cannot.

3.9. Audit Artifacts

The final pipeline produces four classes of audit artifacts:

- **Structural artifacts**: sectioned, patched, captioned, and chunked JSON files with source identity and breadcrumbs.
- **Index artifacts**: Qdrant collections with named dense, sparse, and page-image vectors plus source payloads.
- **Retrieval artifacts**: per-question contexts, final output payloads, selected chunks, token estimates, and visual-route diagnostics.
- **Answering artifacts**: raw Qwen responses, usage records, latency records, three-pass Codex review scores, and review notes.

These artifacts are needed for validity. They allow a low-scoring answer to be classified as a retrieval failure, answering failure, mixed failure, over-context failure, unsupported answer, visual hurt case, or truncation failure. Without them, the experiments would only report aggregate scores, which would not explain why a configuration should or should not become a final preset.

4 | Research Methodology

4.1. Core Phases and V3 Robustness Check

The experiments are built from two core phases and one later robustness check. The first phase, archived as v1, provides the initial system baseline: it shows that the pipeline can parse a real technical manual, repair missing icons, caption screenshots, build text and visual indexes, retrieve evidence, and answer a reviewed question set. It also identified several directions that should not become defaults, including rendered-page answering, thinking-on, and high-cost visual retrieval. The second phase, v2, does not repeat every v1 experiment. It keeps the v1 quality baselines where the earlier evidence was already strong, tightens the stage artifact contracts, and performs the final multi-document parameter study using real downstream answers and three-pass Codex review. The third phase, v3, is narrower: it keeps the v2 pipeline fixed and tests whether text-retrieval backbone changes can explain the remaining answer-quality weakness.

Phase	Core question	Evaluation material	Role in the thesis
v1 baseline	Can a multimodal manual-RAG pipeline be built, measured, and reproduced end to end?	One main technical-manual pipeline; 220 reviewed retrieval questions; 50 answering questions; visual and thinking appendix checks.	Establishes the runnable baseline and determines the parsing, patching, captioning, and initial indexing settings inherited by v2.
v2 final validation	Which chunking, retrieval, and answering presets should be submitted as the final system for multi-document technical-manual QA?	14 source PDFs; 50-question parameter search; 200-question final validation; 88 answer-bearing runs; 6001 generated answers.	Provides the main answer-centered evidence for final presets, rejected modes, cost trade-offs, and latency trade-offs.
v3 retrieval-backbone check	Are the remaining v2 weaknesses mainly caused by the text-retrieval backbone?	Fixed v2 corpus, chunks, prompt, and answerer; 50-question screening; 200-question confirmation; matched v2-high control.	Serves as a diagnostic robustness check showing that dense/sparse backbone substitution alone cannot close the direct-PDF gap.

Table 4.1: Research roles of v1, v2, and v3

The full corpus details, scoring inputs, fixed baselines, and varied parameters are preserved in the appendix detailed archive table; see table A.2.

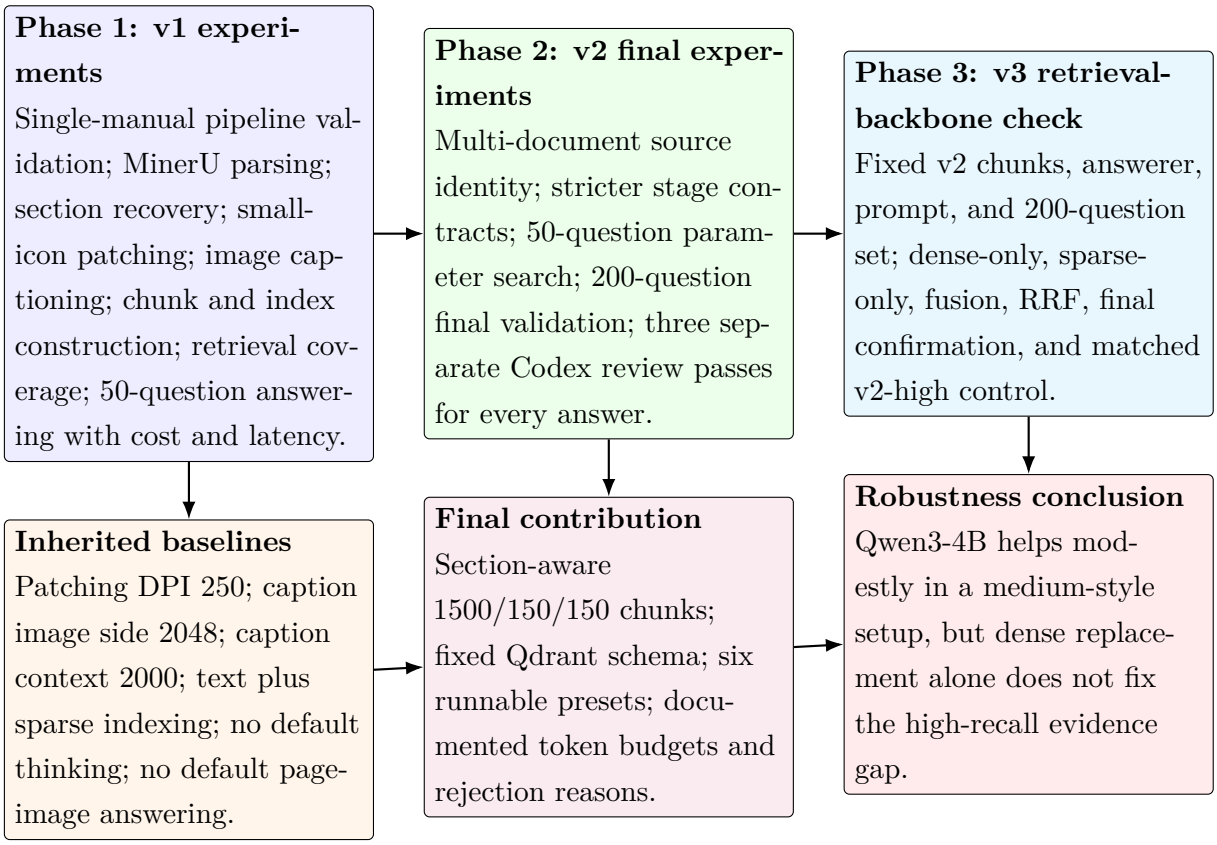


Figure 4.1: Relationship among v1 experiments, v2 final validation, and v3 robustness checks

The three phases answer different questions. v1 asks whether a multimodal manual-RAG pipeline can be built and measured end to end. v2 decides which contracts and presets belong in the final system. v3 is narrower: it tests whether the observed weakness is mostly a consequence of the original dense/sparse retrieval backbone.

The organization follows the actual iteration path. Ignoring v1 would remove the design evidence explaining why patching, captioning, thinking, and page-image answering are not retested from scratch in v2. Treating v1 as final would miss the stricter multi-document scoring in v2. V3 prevents another misreading: attributing all v2 weakness to a single embedding stack. The study uses v1 to explain the starting point, v2 to make preset decisions, and v3 to bound the effect of text-backbone substitution.

4.2. Why the Study Is Sequential Instead of Full Factorial

The full parameter space is too large for a complete Cartesian ablation. A literal full-factorial design across chunk mode, chunk size, overlap, minimum size, retrieval route, visual bonus, visual weight, retrieval `k`, final `top_k`, context cap, score ratio, RRF constant, output token budget, thinking, image input, dense model, dense vector size, sparse model, and sparse-fusion setting would require thousands of complete answer runs. Each complete v2/v3 run requires Qdrant retrieval, Qwen answering, and Codex review passes for every question. This would be expensive while also testing many nonsensical combinations, such as visual-bonus settings without visual hits or output-token sweeps under unstable retrieval.

For these reasons, the thesis uses a controlled sequential methodology:

1. freeze upstream stages before testing downstream stages;
2. search one parameter family at a time under a fixed, documented baseline;
3. preserve representative rejected configurations for interpretation;
4. validate deployable full presets on the 200-question set;
5. run narrow robustness checks only after the final v2 baseline is known;
6. choose defaults by a Pareto rule, not by maximum mean score alone.

The sequential design trades full-factorial completeness for a traceable experimental chain. Preset decisions are supported by completed runs, and the one derived image-input counterpart is marked as derived rather than treated as a separately validated 200-question row. Unexecuted full-factorial searches are recorded as scope boundaries; rejected modes are kept with concrete experimental evidence. This compromise matches the actual cost constraints of the project: every answer-scored configuration requires generation plus three review passes.

The cost of a full factorial design can be estimated from the variables that were actually considered. Even a simplified grid with 8 chunk sizes, 6 context caps, 5 retrieval-`k` values, 7 final-`top_k` values, 7 RRF constants, 6 score-ratio values, 3 routes, 5 visual weights, 5 output-token budgets, 2 thinking settings, 2 image-input settings, 7 dense candidates, and 4 sparse candidates would contain tens of millions of combinations before repeated review. Most of these combinations would be meaningless or redundant. For example, visual weight has no effect when the route is text-only; image input does not change re-

trieval ranking; output-token sweeps should not be mixed with unstable retrieval settings; chunking changes require reindexing; and dense-vector-size changes require a different Qdrant schema.

Decision point	Why it is tested first or separately	Downstream dependency	Final claim boundary
Patching DPI and captioning context	These alter the textual evidence created during ingestion	All later chunks and indexes depend on enriched content	v1 baselines are inherited; v2 tightens contracts.
Chunking profile	It changes chunk ids, text, context shape, and index payloads	Text index must be rebuilt for every profile	15 answer-scored profiles, not a full grid.
Retrieval settings	They change evidence ordering and prompt length without changing the index	Answering can be rerun without reindexing	Broad sequential search plus interaction checks.
Text retrieval backbone	It changes dense/sparse vectors and sometimes vector size	A new text index and sometimes a new collection schema are required	Tested only in v3 after the v2 answer-centered baseline exists.
Answering settings	They change generation behavior and image payloads	Retrieval output is fixed for feature tests	Output budget, thinking, and image input are tested directly.
Final presets	They combine deployable settings into named modes	Validated on 200 questions	Presets are supported under the measured serving environment.

Table 4.2: Sequential experiment logic

The ordering prevents downstream comparisons from being made on stale or unstable upstream artifacts.

4.3. V1 Methodology

The first version used stage-specific experiments because the pipeline did not yet have a stable end-to-end scoring contract. Each stage was evaluated with the most appropriate observable at the time.

Stage	Main object	Main measurements	Methodological role
Parsing and sectioning	MinerU outputs and PDF outline	Existence of content JSON, origin PDF, images, outline mapping	Establishes a structured input for all downstream stages.
Patching	Text/list/table fields and icon crops	4483 checked fields in the full run; 72 target icons and 150 reviewed fields in quality review; fields/min and p95 latency	Selects the small-icon repair baseline and identifies patching as the main runtime bottleneck.
Captioning	285 true image blocks	Image descriptions, images/min, p95 latency, effect of image side and context tokens	Selects the image-captioning baseline used by v2.
Chunking and indexing	Enhanced document chunks and Qdrant collection	Chunk count, chunk metadata, dense/sparse vectors, page-image visual vectors	Creates the first retrieval substrate.
Retrieval	220 reviewed questions	Any-gold MRR, Recall@10, primary hit, gold coverage, context size, latency	Measures whether evidence can be retrieved before testing final answering.
Answering	50 answering questions and appendix hard subsets	Main 14 configurations x 50 questions = 700 calls; 148 appendix calls for thinking and ColPali checks; score, usable rate, tokens, latency	Tests whether retrieved evidence produces usable answers and whether page images or thinking should become defaults.

Table 4.3: V1 methodology by stage

V1 combines quality review, throughput measurement, retrieval coverage, and answering checks, because the first system did not yet have a single final scoring protocol.

The v1 answering set was deliberately mixed: 16 visual image/UI/caption questions, 14 visual text or operation questions, 10 table-evidence questions, and 10 nonvisual controls. That composition kept the first experimental phase focused on the technical-manual problem rather than on generic text-only RAG.

The v1 methodology also introduced the habit of separating quality review from throughput review. For patching, DPI was evaluated by visual quality and text preservation, while concurrency and batch size were evaluated by throughput and p95 latency after the quality setting had been chosen. For captioning, image side and context length were treated as quality-relevant inputs, while concurrency and checkpoint interval were treated as serving parameters. This separation becomes a formal principle in v2.

4.4. V2 Methodology

The second version changes the unit of evidence from a single manual prototype to a multi-document indexed corpus. The v2 source set contains 14 PDFs and a final Qdrant collection named `rag-flow-v2`. Every relevant block, chunk, retrieval hit, and final answer

carries stable source identity so that evidence can be audited across files.

The v2 experimental loop is the same across chunking, retrieval, answering, and final preset validation.

Algorithm 4.1 V2 answer-centered evaluation loop

1: Select one configuration family and fix all upstream stages.
2: Run the pipeline stage required by the configuration.
3: **if** chunk text or chunk ids changed **then**
4: Rebuild the affected Qdrant text index.
5: **end if**
6: **if** visual route or image input requires page vectors **then**
7: Use the fixed page-level ColPali visual index.
8: **end if**
9: **for** each question in the evaluation set **do**
10: Retrieve final evidence payload under the current configuration.
11: Send the final payload to Qwen/SGLang with the fixed answer prompt.
12: Store retrieved context, final output payload, answer, usage tokens, and latency.
13: Review the answer three times against the canonical answer, gold-evidence summary, and retrieved-context excerpt.
14: **end for**
15: Aggregate mean score, high-quality rate, low-score rate, context tokens, total tokens, efficiency, latency, and failure types.

The evaluation loop is answer-centered: retrieval configurations are not selected only because they retrieve a gold chunk; they are selected because the final generated answer improves under an explicit token and latency budget. This matters for technical manuals, where a retrieved page can contain the correct answer but still overwhelm the answerer with neighboring procedures or unrelated UI descriptions.

The loop also enforces a clean boundary between configuration families. Chunking experiments rebuild the text index because chunks and chunk ids change. Retrieval experiments reuse a fixed index because they change only query-time selection. Answering experiments reuse retrieval outputs or fixed retrieval settings because they change the generation payload. This prevents a common experimental mistake: attributing a score change to retrieval when it was actually caused by stale chunks, or attributing an answering improvement to image input when the image paths were not actually delivered.

Experimental artifact	Stored content	Why it matters
Run manifest	Parameters, run id, source paths, model and service settings	Makes every score traceable to a concrete configuration.
Retrieval context	Selected chunks, ranks, scores, context text, estimated tokens	Allows low-score answers to be classified as retrieval or answering failures.
Final output payload	Text sent to the answerer and optional image URLs	Verifies what Qwen actually received.
Raw answer	Qwen response, usage tokens, latency	Separates generation behavior from review behavior.
Review passes	Three pass scores and notes per answer	Reduces single-pass variance and preserves adjudication evidence.

Table 4.4: Run artifacts saved for answer-centered evaluation

These artifacts make the experiments inspectable: a score can be traced back to the retrieved context and the exact answer that was reviewed.

4.5. V3 Robustness-Check Methodology

The v3 experiment is not a new end-to-end pipeline design. It is a controlled robustness check for one specific interpretation: whether the weak final RAG answer quality is mainly caused by the selected text-retrieval backbone.

The v3 runs keep the final v2 corpus, sectioned/patched/captioned JSON, section-aware `1500/150/150` chunks, answer prompt, answer model, visual setting, image-input setting, and thinking setting fixed. They vary dense model, dense vector size, sparse model, dense/sparse fusion, and RRF constant. Dense-vector-size changes require a separate Qdrant collection because the dense vector schema changes. Sparse modifier is not treated as an experimental variable; the current Qdrant `IDF` sparse-vector policy is recorded as part of the schema.

The selection process has four steps. Phase 1 compares dense-only and sparse-only candidates on the 50-question search set. Phase 2 fuses the best single-route candidates with the current routes. Phase 3 performs a small RRF sweep only for the most promising fused candidate. Phase 4 confirms the selected candidate on the full 200-question set. A

later matched v2-high control fixes the original `high` budget and changes only the dense model, so that the v3 result is not confused with a context-budget or RRF change.

The scoring standard is content-only for v3: the Codex reviewer receives only `question`, `canonical_answer`, and `candidate_answer`. It does not receive retrieved context, gold evidence, source-PDF paths, Qdrant hits, previous scores, or thesis text. This differs from earlier RAG-evidence review rows, but it is intentional: v3 asks whether the final generated answer matches the canonical answer under a controlled answer-side comparison.

For high-memory embedding rows, retrieval and answer generation may be split into two stages. The retrieval stage archives the exact final context for every question, then the embedding model is released and the unchanged Qwen answerer generates answers from those archived contexts. In reporting, the comparable per-question time is retrieval latency plus answer-generation latency. This prevents archived-context runs from appearing faster simply because retrieval was performed earlier.

4.6. Experiment Volume

The final v2 archive records 88 answer-bearing runs in the consolidated summary table. These runs generated 6001 question-answer instances and 18003 separate Codex pass judgments.

Experiment group	Runs	Questions/run	Answer generations	Codex pass judgments
Chunking search	15	50	750	2250
Retrieval sweeps	37	50	1850	5550
Retrieval interactions	13	50	650	1950
Answering token/features	11	50	550	1650
Final 200QA validation	11	200	2200	6600
Smoke run	1	1	1	3
Total	88	–	6001	18003

Table 4.5: V2 experimental volume

The final validation alone contains 2200 full answers and 6600 separate review judgments.

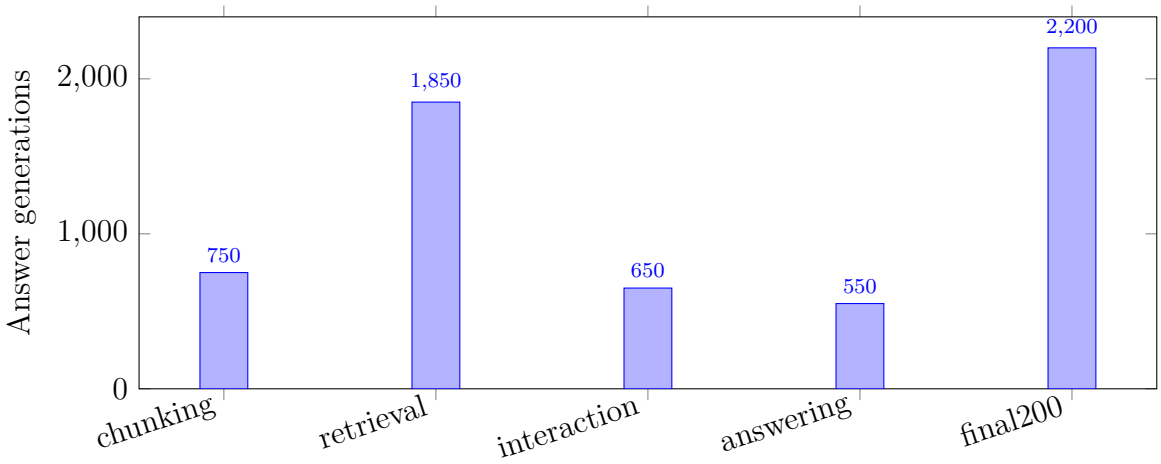


Figure 4.2: V2 answer generations by experiment family

The bar chart compares the number of Qwen answers generated for chunking, retrieval, interaction, answering, and final200 groups, showing that the 200-question final validation is the largest confirmation block.

The later v3 robustness check is intentionally reported separately because it answers a narrower question and uses content-only scoring. It adds 1250 Qwen-generated answers for the main retrieval-backbone experiment, plus the matched v2-high Qwen row and a Codex-over-context diagnostic row.

V3 block	Generated answers	Prompt tokens	Completion tokens	Total Qwen tokens	Avg reported latency
Main retrieval-backbone check	1250	12,211,348	147,150	12,358,498	4.30 s
Matched v2-high Qwen row	200	3,429,159	27,452	3,456,611	9.76 s
Matched v2-high Codex-over-context row	200	n/a	n/a	n/a	13.58 s

Table 4.6: Additional v3 robustness-check volume

Codex token usage is unavailable for the v3 reviewer and Codex-over-context calls because the CLI does not provide complete prompt and completion accounting. Timing is interpreted at run-family level: archived-context diagnostic rows add retrieval latency back to answer-generation latency, while aggregate Qwen token accounting comes from the answer-generation logs.

The archive is intentionally larger than the final set of tables in the main text. Some runs are direct candidates for deployment, while others are stress tests or rejected controls. The 24k context run, thinking-on run, repaired image-input run, and bbox visual route are all examples of runs whose main value is diagnostic. Including them prevents the main text from only reporting winning configurations; it shows why tempting alternatives

were rejected.

Run class	Count	Role in final argument
Winner or near-winner runs	9	Provide candidates for final presets and Pareto comparison.
Rejected stress cases	6	Demonstrate serving limits, over-context behavior, thinking latency, and bbox instability.
Parameter-family sweeps	61	Map the main search surface around chunking, retrieval, and answering.
Interaction checks	11	Check whether single-parameter conclusions survive deployable combinations.
Smoke run	1	Confirms pipeline wiring but is not used for quality conclusions.

Table 4.7: Interpretive role of the v2 run archive

The archive supports both preset selection and rejection decisions; the rejected rows are kept because they explain why tempting modes were not promoted.

4.7. Data Provenance and Reproducibility

The thesis separates source evidence from system artifacts. Canonical answers and gold evidence notes are built from original PDFs, rendered pages, crops, and document text. MinerU JSON, patched JSON, captioned JSON, chunks, and Qdrant payloads are experimental artifacts used to test the system, not the source of truth for the benchmark.

Every major run is traceable through a run id. For v2, the consolidated table records the run id, route, context cap, retrieval `k`, final `top_k`, score ratio, RRF constant, thinking flag, image flag, query count, mean score, token statistics, latency, and index timing where applicable. For v1, benchmark CSV files record patching quality, patching throughput, captioning throughput, retrieval metrics, answering summaries, and appendix checks. For v3, row manifests and summary tables record dense model, dense vector size, sparse model, sparse modifier policy, collection size, indexing time, retrieval latency, answer tokens, and the content-only reviewer score. This provenance allows the thesis to combine v1, v2, and v3 without treating older or later results as anecdotal background.

The public implementation is available in the RAG-Flow GitHub repository [1]. The repository is used as the code-availability reference for the pipeline implementation. The numerical results in this thesis remain tied to the archived run manifests, CSV tables,

JSON artifacts, and review files described in the appendix, because source PDFs, model weights, Qdrant collections, private runtime caches, and large run artifacts are not assumed to be part of the public repository.

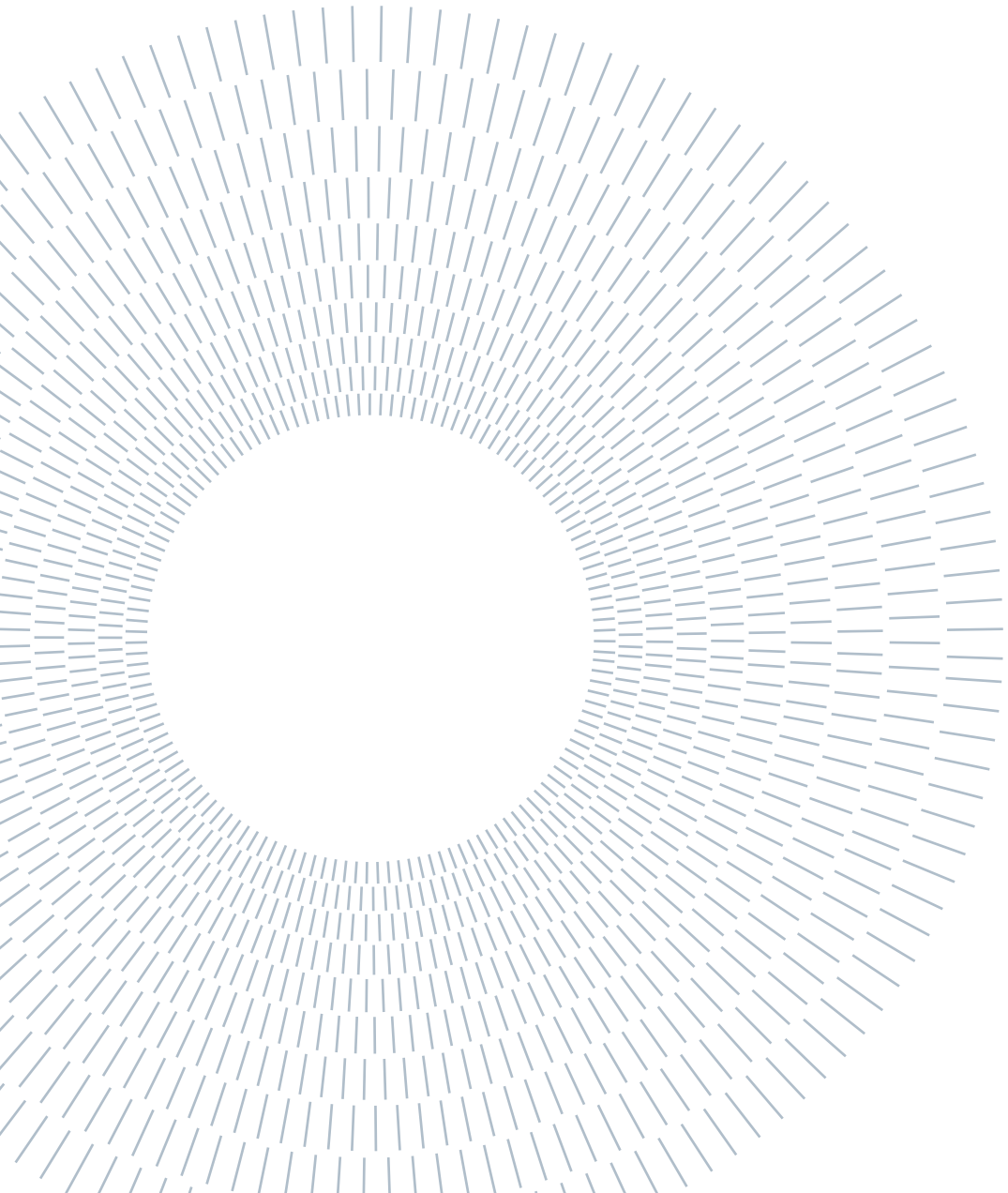
The experimental machine is also part of the provenance record, especially for latency, throughput, and serving-boundary claims. The first full environment snapshot was recorded in the v1 environment archive on 2026-05-04 and is carried into this thesis in table 4.8. Later v2 and v3 answer-generation rows use the same RAG-Flow artifact contract and Qwen/SGLang answer-serving family, while their run manifests record the model, route, token budget, latency, and context cap actually used. Hardware-dependent quantities such as concurrency, GPU memory pressure, and wall-clock latency are measurements under this class of remote GPU serving environment, not hardware-independent algorithmic constants.

Component	Recorded configuration	Reproducibility role
Host type	AutoDL/SeeTaCloud GPU container	Remote GPU environment used for the first complete pipeline validation and later serving-style experiments.
Operating system	Ubuntu 22.04.5 LTS; Linux 5.15	Base system for MinerU, Qdrant, SGLang, and Python environments.
GPU	NVIDIA GeForce RTX 5090; 32607 MiB VRAM; driver 580.105.08; CUDA 13.0	Determines feasible VLM serving, ColPali indexing, Qwen/SGLang answering, concurrency, and memory pressure.
CPU and RAM	Intel Xeon Platinum 8470Q; 208 logical CPUs visible; cgroup memory limit 96636764160 bytes, about 90 GiB	CPU count affects preprocessing throughput; the cgroup memory limit is the effective RAM budget even though `free -h` shows a larger host memory view.
Storage	Root disk about 30 GB; data disk `/root/autodl-tmp` about 250 GB	Large models, caches, parsed PDFs, Qdrant collections, and run artifacts are placed on the data disk rather than the system disk.
Directory layout	Code under `/root/RAG-Flow`; runtime/cache/model data under `/root/autodl-tmp`; `/root/.cache` and `/root/.local` linked to the data disk	Keeps model caches and private configuration outside the small system partition.
Python environments	Separate MinerU, pipeline, and LLM environments; MinerU validation used Python 3.12.13, `mineru 3.0.9`, `torch 2.9.0`, and `pypdfium2 4.30.0`	Avoids dependency conflicts among PDF parsing, indexing, and model serving.
Serving stack	Local OpenAI-compatible SGLang service for Qwen answering/VLM calls; Qdrant text and optional visual collections	Defines the online answering boundary used by v2 and v3 RAG runs. The active serving context limit explains why 24k retrieved context is treated as an over-context failure.

Table 4.8: Experimental machine and serving environment snapshot

The thesis uses this provenance in two ways. First, numerical claims in tables are backed

by CSV or JSON artifacts rather than by manual transcription from screenshots. Second, qualitative claims such as “image input did not help” or “24k context fails operationally” are backed by run-level diagnostics: image URL counts, usage-missing cases, score-bucket distributions, and failure notes. This keeps the thesis grounded in repeatable artifacts while still allowing interpretation.



5 | Evaluation Protocol

5.1. Evaluation Objects

The final experiments score complete answers, not retrieval hits alone. For each configuration, the system runs a real Qwen answer using the retrieved evidence. Codex 5.5 with extra-high reasoning then scores the answer from a prompt containing the question, canonical answer, system answer, compact gold-evidence summary, and a retrieved-context excerpt clipped by the scoring script. If an answer is correct only because of model prior knowledge but is not supported by that excerpt, it does not receive a high score. Usage tokens, latency, retrieval paths, returned chunks, and configuration are stored with the run for analysis rather than supplied as reviewer prompt fields.

Compared with the v1 automatic coverage score, this protocol is stricter. v1 used automatic must-include coverage to compare early answering configurations quickly. The final v2 protocol uses three separate Codex 5.5 extra-high passes per answer and averages the integer pass scores.

The protocol is designed around the failure modes of manual QA. A generated answer can sound fluent while mixing values from two table rows, skipping a required prerequisite step, or citing a UI option from the wrong page. Conversely, a retrieval context can contain the right evidence but still be too noisy for the answerer to use. For this reason, the main metric in the thesis is the reviewed complete answer rather than retrieval hit-rate alone. Retrieval hit-rate, context length, image count, and latency remain necessary diagnostic fields, but none of them alone determines the final preset.

5.2. Question Sets

The second-version experiments use two question sets.

- **50-question search set.** This is the quick parameter-search set. It covers textual explanations, tables, operation steps, visual/UI evidence, and multi-page or multi-chunk questions.

- **200-question final set.** This is the final validation set. It is not used for broad parameter search. It checks whether the 50-question conclusions survive a larger and more varied evaluation.

The gold answers and evidence notes are built from original source PDFs, rendered pages, crops, and document text. They are not generated from the chunked JSON. Chunk ids are diagnostic and may change when chunking changes; source pages and evidence notes remain the stable ground truth.

The 200-question set was built with an LLM-assisted but source-grounded workflow. Codex and ChatGPT were used to help draft candidate manual questions, expected answers, and evidence notes from the PDF corpus. The author then manually checked those candidates against original PDFs, rendered pages, and source text before they were used as benchmark items. This workflow was chosen because the target task is realistic industrial manual support: the questions should resemble what a user may ask about functions, procedures, specifications, diagrams, and compatibility, while the canonical answers still have to be anchored to source pages. The LLMs accelerate question drafting, but the benchmark ground truth is the reviewed PDF evidence rather than the model-generated draft itself.

The 50-question set is used only as a fast search instrument over the same reviewed evidence pool. It is allowed to guide parameter selection, but it is not treated as final evidence. The 200-question set is kept for confirmation so that a setting selected on the smaller sweep must still survive a broader distribution of sources, question types, and evidence layouts.

Set	Use	Guardrail
50-question search set	Fast parameter search for chunking, retrieval, and answering families	May select candidates, but cannot by itself justify a final deployable preset.
200-question final set	Confirmation of representative full configurations and final presets	Not used for broad exploratory search; prevents overfitting the 50-question set.
Original PDFs and rendered pages	Source of canonical answers and visual evidence	Treated as ground truth even when extracted chunks change.
Mapped chunks	Diagnostic mapping from evidence pages to current chunks	Used for failure analysis, not as the answer-quality score.

Table 5.1: Evaluation split and guardrails

The split keeps the search stage small enough to run repeatedly while preserving a larger confirmation layer for final claims.

5.3. Source Corpus and Gold Evidence

The final v2 corpus contains 14 source PDFs and 1114 pages. The corpus is intentionally mixed: long enterprise-software manuals dominate the page count, while short camera and switch sheets contribute visually dense diagrams, datasheets, installation pages, and port-layout references.

Source family	PDFs	Pages	Role in the benchmark
DSS Ultimate manuals	3	1037	Long software manuals with procedures, tables, UI screenshots, and cross-page workflows.
HAC/HDCVI camera documents	6	55	Datasheets, installation sheets, camera manuals, dimension drawings, notices, and visual accessory references.
S3006 switch documents	5	22	Compact datasheets, dimension drawings, installation diagrams, port diagrams, and switch manuals.
Total	14	1114	Multi-document technical-manual corpus used for the final v2 experiments.

Table 5.2: Final v2 source corpus

Page counts are measured from the source PDFs.

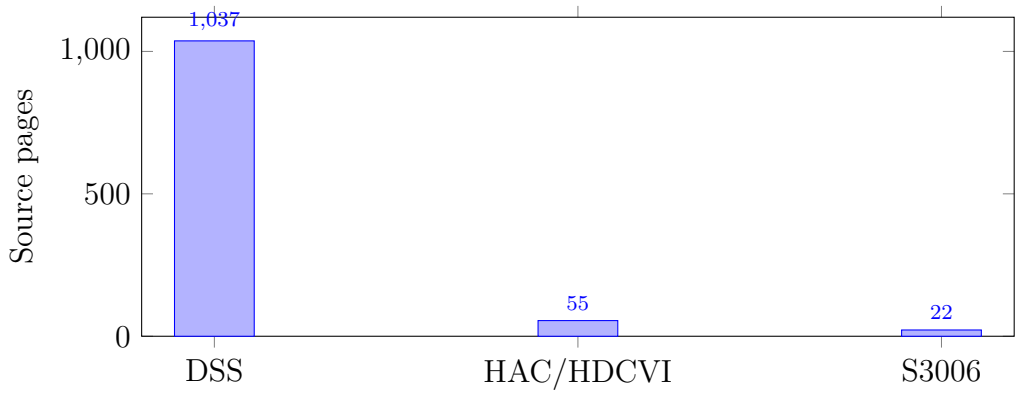


Figure 5.1: Source-page distribution of the final v2 corpus

The page distribution matters because long DSS manuals dominate text volume, while the shorter camera and switch documents contain the densest visual evidence.

The 200-question gold set contains 244 evidence items. It includes 34 questions marked with `requires_visual=true`, 44 multi-evidence questions, and 15 multi-PDF questions. A Codex 5.5 extra-high review pass checked the gold set against the original source PDFs and rendered visual pages before the final experiments. Two kinds of corrections were made during that review: visual accessory labels that had been reversed in one installation sheet were corrected, and several loose support notes were tightened to point at the exact

evidence needed for the answer. This review step is why the evaluation treats original PDFs and rendered pages as ground truth rather than the system's extracted JSON.

The corpus deliberately includes documents with very different extraction profiles. The long DSS manuals provide many procedures and UI screenshots with dense text; they test whether retrieval can find the right operation among many similar operations. The camera and switch sheets are short but visually concentrated; some pages contain almost no extractable text, so the pipeline must preserve page identity and visual evidence even when text retrieval alone is weak. This mix is why the final presets include both a text-only default and explicit visual/image modes.

PDF id	Document type	Pages	Target Qs	TOC entries	Text chars
`dss_pc_manual`	PC client user manual	540	58	634	852777
`dss_web_manual`	Web client user manual	447	48	538	712187
`dss_quick_deployment`	Quick deployment manual	50	15	42	70007
`hdcvi_camera_manual`	Camera user manual	45	21	56	45976
`switch_user_manual`	Switch user manual	16	13	18	20460
`s3006_datasheet`	Switch datasheet	3	8	8	4294
`hac_hf3805g_datasheet`	Camera datasheet	3	8	0	6829
`hac_open_source_notice`	Open-source notice	3	4	0	5647
`hdcvi_box_install_guide`	Visual installation guide	2	5	0	150
`hac_hf3231e_installation`	Mounting compatibility sheet	1	4	0	40
`hac_hf3805g_dimension`	Camera dimension drawing	1	4	0	0
`s3006_dimensions`	Switch dimension drawing	1	4	5	58
`s3006_installation_method`	Installation method sheet	1	4	1	9
`s3006_port_diagram`	Port diagram sheet	1	4	1	0

Table 5.3: Source-PDF inventory used to generate and audit the final 200-question gold set

Short visual sheets deliberately have little or no extractable text, so rendered-page evidence is required.

Review item	Count	Meaning
Source PDFs read into review	14	Every source PDF was included in the gold-set review scope.
Pages read into review	1114	The full corpus page count was available to the review pass.
Total questions	200	Final validation questions after generation and review.
Evidence items	244	Some questions require more than one supporting evidence item.
Text/table evidence verified	208	Evidence checked against original PDF page text.
Visual evidence entries reviewed	36	Visual entries checked against rendered original PDF pages.
Unique rendered visual pages reviewed	7	Rendered-page checks for visual evidence corrections and confirmation.
Remaining needs-review items	0	Review status is pass after corrections.

Table 5.4: Gold-set review audit

The benchmark is anchored to original PDFs and rendered pages, not to whichever chunk artifacts a particular pipeline version happens to produce.

The following table counts the single primary `question_type` assigned to each question.

Question type	Count	Purpose
Fact	41	Direct factual lookup from manual text.
Procedure	45	Ordered operational steps and configuration workflows.
Table or specification	36	Datasheet fields, parameter tables, and specifications.
Visual as primary type	30	Questions whose main category is diagrams, screenshots, drawings, or page-layout evidence.
Cross-page	29	Evidence spread across multiple pages or sections.
Multi-PDF comparison	15	Questions that require selecting or comparing evidence across sources.
Safety or notice	4	License, safety, notice, or warning content.

Table 5.5: Question-type distribution in the 200-question gold set

The two visual-related counts come from different fields in the gold-set JSON. Table 5.5

is a mutually exclusive count of the primary `question_type`; it contains 30 rows where `question_type=visual`. The larger count, 34, comes from the separate boolean field `requires_visual`. The four extra rows are multi-PDF comparison questions whose main task is source comparison, but whose answer still depends on diagram evidence.

5.4. Three-Pass Codex Review

The answer archive is too large for a full blinded human review of every generated answer: the consolidated v2 runs contain 6001 generated answers, and the three-pass protocol produces 18003 individual pass judgments. Codex 5.5 with extra-high reasoning is therefore used as a high-capability external reviewer to reduce evaluation time, apply a consistent rubric, and save auditable review notes for each answer. This choice is practical rather than absolute: Codex review is treated as scalable diagnostic evidence, not as unquestionable human ground truth.

Each answer is reviewed three times by Codex 5.5 with extra-high reasoning. The passes use the same rubric and stored scoring inputs but do not read the previous pass's score or notes. Each pass assigns an integer from 0 to 5. The final answer score is the arithmetic mean, so final scores can be fractional. Review notes are saved for every pass and summarize the key omissions, unsupported claims, or evidence mismatch.

The thesis reports this as an automated Codex review protocol rather than a blinded human panel. In this setting, "independent" means that the three passes are separate reviewer invocations with no access to each other's score or notes.

The review unit is one answer to one question under one configuration. The reviewer is instructed to compare the answer with the canonical answer, then check whether the retrieved-context excerpt supports it. The reviewer does not inspect the original PDFs or the complete retrieval artifacts during this scoring pass. Visual evidence affects the score only insofar as it appears in retrieved text, captions, image-policy metadata, or the answer content. The review should not reward an answer only because it is plausible. It should also not punish harmless wording differences when the required facts and conditions are present. This makes the score closer to a practical user-facing quality judgment than to a literal string match.

Score	Meaning
5	Fully correct, covers the canonical answer, and is supported by the retrieved-context excerpt.
4	Mostly correct, with minor omissions that would not mislead the user.
3	Partially correct, but missing important steps, constraints, fields, or multi-evidence synthesis.
2	Weakly related and not reliable enough to deliver as an answer.
1	Mostly wrong or based on irrelevant evidence.
0	Empty answer, severe hallucination, refusal, or response unsupported by the retrieved-context excerpt.

Table 5.6: Final 0–5 answer-quality rubric

Review input	Purpose
Question and canonical answer	Define the expected content and required conditions.
System answer	Object being scored.
Gold-evidence summary	Gives compact source support for the expected answer.
Retrieved-context excerpt	Shows the clipped evidence text exposed by the scoring script; the default limit is 6000 characters.
Query id	Joins pass-level scores back to the run artifacts.

Table 5.7: Inputs available to each Codex review pass

Usage tokens, latency, returned chunk ids, image URL counts, retrieval files, and configuration fields are stored in the run artifacts and used for aggregate analysis. They are not included as direct text fields in each Codex scoring item.

The review judges excerpt-supported answer quality rather than textual similarity alone.

The three-pass design treats Codex review as a controlled, repeatable scoring protocol rather than as a perfect evaluator. It is a practical compromise between single automatic scoring and a full human panel. Because all pass-level notes are saved, the thesis can inspect disagreements and failure causes. The final interpretation combines aggregate scores with failure analysis, score buckets, and run-level diagnostics.

After the aggregate tables were produced, the author manually spot-checked representative final-200 rows from the archived runs. This check read the question, canonical

answer, gold-evidence summary, generated answer, pass-level review notes, and stored run metadata such as context size, image URL count, and failure label. It was not a second blinded annotation campaign and it was not used to overwrite any score. Its purpose was narrower: to verify that the main score differences corresponded to recognizable retrieval, answering, or image-handling mechanisms, and to identify cases where an individual automated score should be interpreted with caution. The qualitative examples in Section 9.9 therefore explain the quantitative results rather than replacing them.

5.5. Score Families and Comparability

Later chapters report both final preset scores and diagnostic rescores, so the thesis uses two score families that are not numerically interchangeable. The main v2 preset selection uses the RAG-evidence review described above: the reviewer receives the question, canonical answer, candidate answer, compact gold-evidence summary, and clipped retrieved-context excerpt. This score asks whether the deployed RAG system produced an answer that is both correct and supported by the retrieved evidence excerpt exposed to the reviewer.

The later direct-PDF diagnostic uses an additional content-only rescore. In that diagnostic, the reviewer receives only the question, the canonical answer, and the candidate answer. It does not receive retrieved context, gold evidence, source-PDF paths, or retrieval metadata. This score asks a narrower question: how correct is the candidate answer as text, under the same answer-only reviewer, regardless of how the answer was produced.

Score family	Reviewer inputs	Used for
RAG-evidence v2 score	Question, canonical answer, candidate answer, gold-evidence summary, and retrieved-context excerpt; the default excerpt limit is 6000 characters.	Main chunking, retrieval, answering, and preset selection.
Content-only diagnostic rescore	Question, canonical answer, and candidate answer only. No retrieved context, gold evidence, source-PDF paths, or retrieval metadata.	Direct-PDF and context-only diagnostic comparisons only.

Table 5.8: Score families and comparability rules

Numbers from one score family are not directly comparable with numbers from the other family. For example, a preset's RAG-evidence final 200-question score and its later content-only diagnostic score can differ substantially without contradiction, because

the two reviewers are judging different properties. Content-only rows also need to be compared within the same rescore batch unless the text explicitly states that the rows were rescored together. This matters for the later direct-PDF diagnostics: the first direct-PDF table compares selected deployed presets with the direct-PDF answer, while the v3 matched-control table rescored a different five-row comparison set together. Within each table, all rows use the same prompt and are comparable to one another.

RAG-evidence scores are the deployment-selection metric, while content-only scores are diagnostic controls. The RAG-evidence score is an excerpt-supported review; full manual rereading is represented only by the separate direct-PDF diagnostic. The run archive still stores full context files, retrieval JSON, usage, latency, and configuration for audit and failure analysis. Content-only scores answer narrower questions, such as whether a stronger answerer can use the same archived RAG context better, or how much information is still recoverable when Codex inspects the original PDFs directly. A content-only direct-PDF row may score higher because it ignores retrieval support and because it uses a different agentic evidence-access procedure; that result is used as a source-PDF reference point for diagnostic interpretation, while the RAG-evidence scores remain the preset-ranking basis.

5.6. Metrics

The main reported metrics are:

$$\text{mean\_score} = \frac{1}{N}\sum_{i=1}^{N} s_i, \tag{5.1}$$

$$\text{high\_quality\_rate} = \frac{|\{i : s_i \geq 4\}|}{N}, \tag{5.2}$$

$$\text{low\_score\_rate} = \frac{|\{i : s_i \leq 2\}|}{N}. \tag{5.3}$$

Token efficiency is stage-specific:

$$\text{retrieval\_side\_efficiency} = \frac{\text{mean\_score}}{\text{retrieved\_context\_tokens}/1000}, \tag{5.4}$$

$$\text{answering\_side\_efficiency} = \frac{\text{mean\_score}}{\text{total\_LLM\_tokens}/1000}. \tag{5.5}$$

Retrieved-context tokens are estimated from characters using the fixed retrieval constant `RAG_FLOW_RETRIEVAL_CONTEXT_CHARS_PER_TOKEN=4.0`. LLM total tokens come from Qwen/OpenAI-compatible usage records. The two token counts are reported separately.

Keeping these token counts separate matters because retrieval-side and answering-side costs answer different questions. Retrieved-context tokens estimate how much textual evidence the retriever placed into the prompt. Total LLM tokens measure what the serving stack reports after prompt formatting, generation, and optional image handling. Image input can leave retrieved-context tokens nearly unchanged while increasing total tokens. The image-input experiment in this thesis shows that pattern: average context remains close to default, but average total tokens increase sharply when 1378 image URLs are delivered across 200 questions.

The thesis also reports high-quality and low-score rates because the mean score alone can hide failure distribution changes. A configuration that slightly improves mean score but creates more zero or one-point failures may be unacceptable for deployment. Conversely, a compact configuration with a similar mean score and fewer tokens may be preferable even if it does not maximize the high-quality rate.

5.7. Failure Labels and Diagnostic Flags

Each reviewed answer stores a Codex `failure_type` field. In practice, a row can contain one label or a small pipe-separated combination when the reviewer identified more than one cause. The scoring script restricts the stored labels to the following set:

- `retrieval_failure`: necessary evidence did not enter the retrieved context.
- `answering_failure`: evidence entered the context, but Qwen failed to use it correctly.
- `visual_failure`: the missing or misused evidence is mainly visual.
- `pdf_access_failure`: a direct-PDF diagnostic could not access or use the required source PDF evidence.
- `unsupported_answer`: the answer may be true but is not supported by the retrieved evidence.
- `partial_answer`: the answer is useful but omits required facts, steps, conditions, or sources.
- `wrong_answer`: the answer contradicts the canonical answer or selects the wrong evidence.

Other failure evidence is stored as diagnostic fields rather than as `failure_type` labels. These include `usage_missing`, `thinking_leak`, image URL counts, score buckets, and

pairwise comparisons between text, visual-route, and image-input rows. The thesis uses those fields to describe aggregate patterns such as over-context serving failure or visual hurt cases, but those descriptions are not separate scoring labels.

The labels and diagnostic fields are used as an explanatory layer rather than as a second leaderboard. For example, the 24k context run is rejected not only because its mean score is low, but because the run has 124 usage-missing cases and many broken or empty answers under the current context limit. The image-input run is rejected because its additional image evidence does not reduce the main retrieval-failure class enough to justify the token cost. The thinking-on run is rejected because it increases latency and severe failures rather than solving the underlying evidence-selection problem.

Failure evidence	Typical manual-QA example
`retrieval failure`	The answer needs a specific datasheet row, but the retrieved context contains only a neighboring model or generic manual section.
`answering failure`	The relevant row is present, but the answerer selects the wrong column or omits an important condition.
`unsupported answer`	The answer is plausible product knowledge, but the retrieved evidence does not contain it.
`partial answer`	A multi-step procedure is partly answered, but required steps or conditions are missing.
`wrong answer`	The answer selects a neighboring table row, a wrong page, or the wrong source document.
Diagnostic: `usage_missing`	A very large prompt exceeds the stable serving boundary and produces missing usage records or broken answers.
Diagnostic: visual hurt	Visual route or image input changes the evidence mix so that a previously correct answer becomes less grounded.

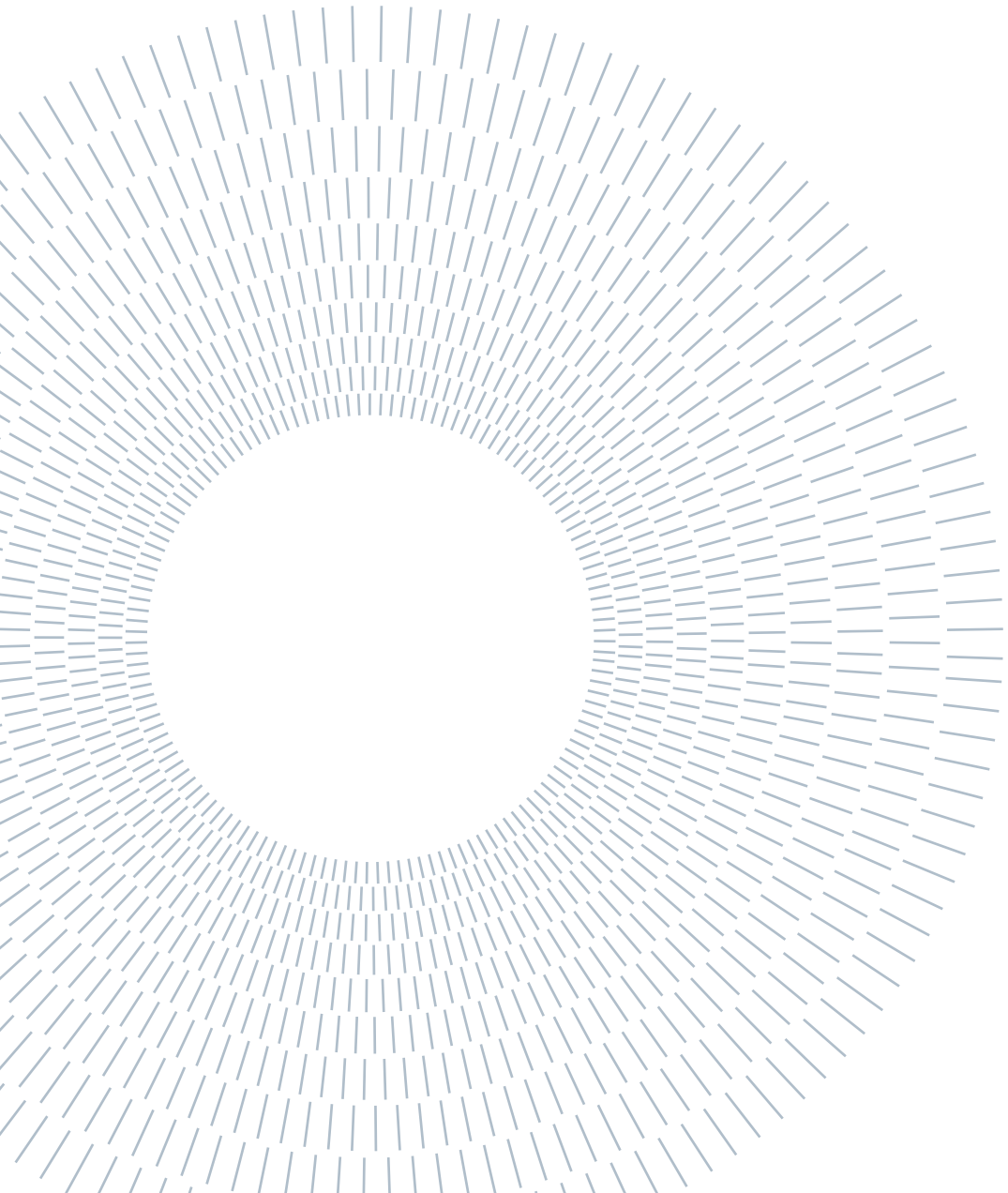
Table 5.9: Examples of failure labels and diagnostic fields in technical-manual QA

The label and diagnostic fields link a low score to a likely repair target: retrieval, answering, context budgeting, visual routing, or serving limits.

5.8. Decision Rule

The default configuration is chosen by a Pareto rule. The highest mean score is not automatically the default. If two configurations are close in score, the one with lower context, lower latency, fewer moving parts, and better token efficiency is preferred. Larger context caps, visual route, image input, or thinking are accepted only when they produce stable gains that justify their operational cost.

The final system therefore provides several presets rather than only one configuration. The best score, the best efficiency, the simplest online behavior, and the best visual recall behavior are not the same point. The preset names encode intended use: `medium` for a conservative interactive baseline, `high` for hard questions, `low` for high-volume or low-token use, `medium-with-visual-recall` for visually distinctive pages, and image-input presets for diagnostics or explicitly visual answers.



6 First-Version Experiments

6.1. Role of the First Version

The first version provides the experimental foundation for the final system. It established that the pipeline could parse a technical manual, repair missing visual symbols, describe image blocks, build text and visual indexes, retrieve evidence, and answer reviewed questions. The final thesis keeps those results as the starting point and uses v2 to refine the contract and final presets.

The v1 work exercised the entire pipeline before the final multi-document evaluation existed. It identified which stages are offline bottlenecks, which visual enhancements affect evidence quality, which serving parameters are only throughput adaptations, and which multimodal ideas look promising in small examples but do not justify default use. These results provide the empirical basis for inherited baselines.

V1 contribution	How it is used in the thesis
End-to-end pipeline evidence	Shows that parsing, sectioning, patching, captioning, chunking, indexing, retrieval, and answering can run as a complete artifact chain.
Patching benchmark	Selects the quality baseline `dpi=250` and identifies patching as the dominant offline runtime stage.
Captioning benchmark	Selects `max_image_side=2048` and `max_context_tokens=2000` for image descriptions.
First retrieval study	Demonstrates strong evidence coverage but also the token cost of large top-k contexts.
First answering study	Shows that text-only answering is stronger than always adding rendered page images.
Thinking and ColPali checks	Motivate thinking-off default and optional `medium-with-visual-recall` rather than default visual retrieval.

Table 6.1: How v1 evidence is reused in the thesis

V2 retests the first-version choices through the final contract and preset validation.

6.2. End-to-End Runtime

The first complete ingest rerun started from existing MinerU outputs and ran sectioning, patching, captioning, and chunking. The run completed successfully in 19 minutes and 2 seconds.

Stage	Time	Share	Main result
Sectioning	0.5 s	0.05%	Recovered 497 PDF outline sections and wrote the sectioned JSON.
Patching	14 m 28 s	76.02%	Checked 4483 fields and patched 1641 fields.
Captioning	4 m 24 s	23.11%	Captioned 285 image candidates.
Chunking	9.3 s	0.81%	Created 543 auto chunks.
Total	19 m 02 s	100.00%	Exit code 0.

Table 6.2: First-version end-to-end timing from sectioning to chunking

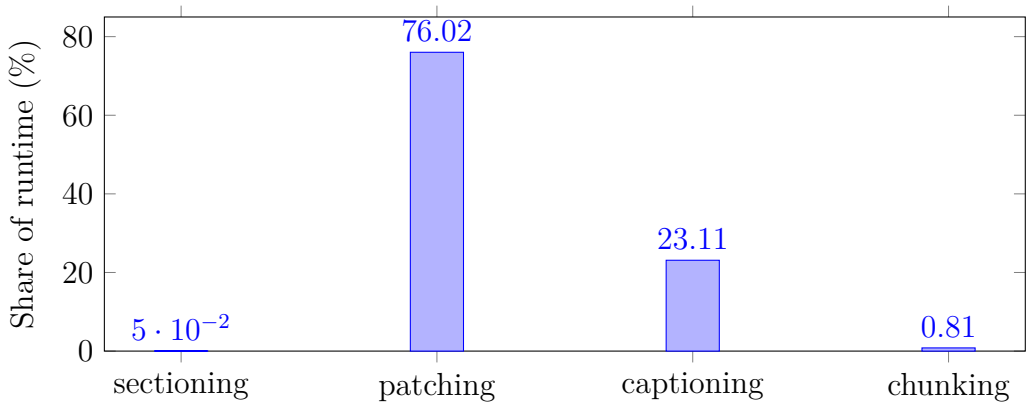


Figure 6.1: Runtime share of v1 offline pipeline stages

The bar chart shows that sectioning and chunking are almost negligible, while small-icon patching dominates runtime and image captioning is the second-largest cost.

The runtime distribution shaped the later research plan. Sectioning and chunking were fast enough that retesting them under multiple parameter settings was feasible. Patching and captioning were expensive enough that their quality settings had to be chosen carefully and then inherited unless there was a strong reason to rerun them. This explains why v2 spends its answer-scored budget mainly on chunking, retrieval, and answering rather than repeating the entire visual-enhancement grid.

The v1 timing also shows why multimodal preprocessing is acceptable in the offline path. Patching and captioning dominate ingest time, but they are paid once when a manual is processed. By contrast, page-image answering and image-input answering are paid every time a user asks a question. This cost asymmetry is a recurring theme in the final recommendations.

6.3. Patching Experiments

The v1 patching experiment compared DPI settings and runtime parameters. The quality review used 72 target icons and 150 reviewed fields. The best quality setting was `dpi=250`: strict icon hit ratio 97.22%, zero missed targets, average quality score 4.74, and text preservation 99.33%.

The experimental design was sequential. DPI was selected first because it changes the visual evidence shown to the VLM and can change quality. Concurrency, batch size, and checkpoint interval were then tuned as serving and throughput parameters under the chosen DPI. This distinction matters for reporting: the final system inherits `dpi=250` as a quality baseline, while concurrency and batch size are treated as deployment parameters that may be adapted to hardware.

Patching targets small inline visual symbols, not general image description. The stage scans fields where missing icons usually change semantics: text blocks, list items, table bodies, table footnotes, and image footnotes. It does not patch headers, footers, page numbers, or most captions, because those fields are either repeated layout material or unlikely to contain inline action icons. The crop shown to the VLM is built from the field bbox and, when available, nearby inline-icon candidates. This local crop policy reduces the chance that the model uses unrelated neighboring text as evidence.

Field group	Patching decision	Reason
`text.text`	Included	Main prose often contains inline button icons and missing UI symbols.
`list.list_items`	Included	Lists often contain action icons or symbol-prefixed steps.
`table.table_body`	Included	Tables can contain icon-only cells or missing status symbols.
`table.table_footnote`	Included	Footnotes can contain click instructions with missing icon glyphs.
`image.image_footnote`	Included	Image footnotes often describe screenshot controls.
Headers, footers, page numbers	Excluded	High repetition and low semantic value for retrieval.

Table 6.3: V1 patching field coverage

The stage targets semantic icon loss while avoiding repeated layout noise.

The quality review was not a full manual-wide icon truth set. It was a targeted visual review over pages selected to contain difficult icon cases: ordinary step icons, table icons, image footnotes, continuous blank placeholders, and dense toolbar pages. Reviewers compared the crop, the original OCR text, and outputs at multiple DPI settings. The main quality fields were strict icon hits, wrong icons, missed targets, false positives, overall patch quality, and whether the original text was preserved.

DPI	Strict hits	Wrong icons	Missed	Strict hit %	Quality score
200	65	6	1	90.28	4.64
250	70	2	0	97.22	4.74
300	68	3	1	94.44	4.63

Table 6.4: Patching quality review

DPI 250 gives the best observed balance of strict icon hits and text preservation.

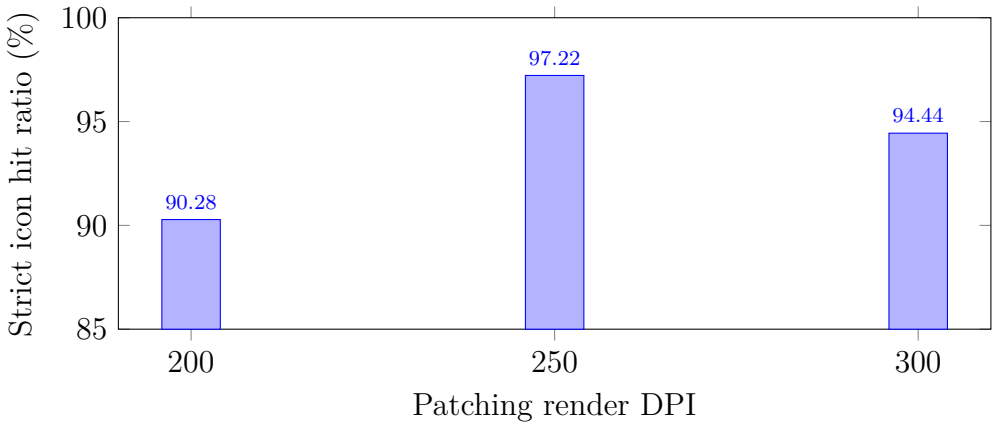


Figure 6.2: Small-icon patching DPI versus strict hit rate

The DPI result illustrates the difference between speed and quality. At the initial speed setting, 200 dpi was faster than 250 dpi and 300 dpi. If throughput were the only metric, 200 dpi would have won. The review table shows why it does not: 200 dpi missed one target and produced more wrong icon outputs, while 250 dpi reached the highest strict-hit ratio and no missed targets. DPI is treated as a quality setting, not a throughput knob.

The throughput sweep showed that increasing concurrency improved fields per minute until roughly 8–10 concurrent requests, after which latency and server pressure grew

without meaningful throughput gains. This supports the final default of `dpi=250` and an environment-dependent concurrency around 8.

Concurrency	Fields/min	p95 request (s)	Max GPU memory (MiB)
1	160.32	0.76	30959
4	279.01	2.10	30967
8	308.73	3.34	30987
10	305.97	4.08	30991
16	306.36	5.88	30991
20	308.01	6.94	30991

Table 6.5: Representative v1 patching throughput results

Concurrency mostly affects serving stability and throughput, not the quality baseline.

Parameter sweep	Setting	Throughput	Interpretation
Batch size	8	227.56 fields/min	Client scheduling bottleneck.
Batch size	32	255.38 fields/min	Better, but still below plateau.
Batch size	140	296.87 fields/min	Near stable high-throughput region.
Batch size	256	300.06 fields/min	Similar to 512, more conservative memory footprint.
Batch size	512	309.61 fields/min	Highest observed throughput.
Checkpoint interval	1	310.70 fields/min	Best recovery granularity with tiny write overhead.
Checkpoint interval	2	312.43 fields/min	Highest observed but not materially different.
Checkpoint interval	30	307.33 fields/min	Older default; worse recovery risk.

Table 6.6: Additional patching runtime sweeps

Batch size mainly reduces client scheduling overhead; checkpoint interval is selected for recovery safety because throughput differences are small.

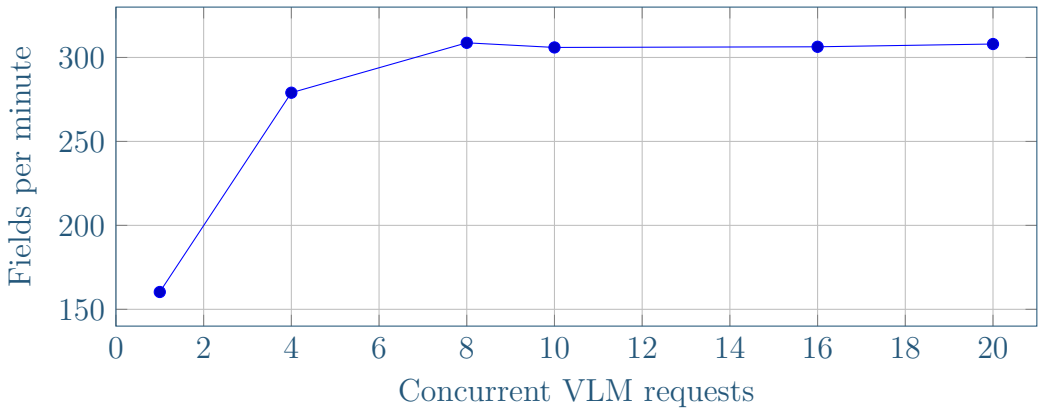


Figure 6.3: Small-icon patching concurrency versus throughput

The final patching configuration was repeated three times under the selected settings. Throughput was 304.59, 304.45, and 313.38 fields/min; p95 latency was 3.54, 3.47, and 3.46 seconds. The repeatability matters because the patching stage is the dominant offline runtime cost. A parameter that only wins once due to queue noise would not be a safe baseline. The repeated result supports using `dpi=250`, `batch_size=512`, and `checkpoint_interval=1` as the v1 baseline, with concurrency tuned around the serving plateau.

The patching benchmark also shaped the v2 safety contract. Some LLM patch outputs can include explanation text, malformed icon names, or only an icon marker without the original OCR text. V1 already used fallback and rejection behavior to preserve text. V2 formalizes that lesson: no-missing output must be explicit, malformed markers are rejected, and only-icon outputs are merged back into the original text rather than replacing it.

6.4. Captioning Experiments

The v1 captioning benchmark captioned 285 image candidates. It selected a maximum image side of 2048 pixels and a 2000-token context budget as the quality baseline. Context length had a large effect on throughput: 2000 context tokens achieved 29.11 images/min, while 10000 tokens dropped to 19.73 images/min.

Captioning differs from patching in both scope and output. It processes true image blocks such as screenshots, diagrams, architecture drawings, figures, and charts. It does not process small inline icons, because those have already been normalized as text by patching. For every true image block, captioning writes a description into `image_description_vlm` and also records an image-answering policy. That policy later allows retrieval to decide whether the original image should be available to a multimodal answerer.

The captioning experiment used the full set of 285 candidates rather than a small page window because image blocks are sparser than patching fields and their distribution is uneven. It completed 31 benchmark runs and 8835 captioning requests with no request-level failures in the reported benchmark. The main experimental variables were image side, context-token budget, concurrency, batch size, and checkpoint interval.

Max image side	Context tokens	Concurrency	Images/min	p95 request (s)
1024	10000	1	19.93	4.41
1536	10000	1	19.64	4.44
2048	10000	1	19.75	4.41
2048	0	1	28.36	3.31
2048	2000	1	29.11	3.32
2048	5000	1	26.16	3.50

Table 6.7: Representative captioning throughput results

The inherited quality baseline keeps 2048 image side and 2000 context tokens.

Figure 6.4: Image-captioning throughput for single-thread and recommended serving settings

The image-side result shows that the model was sensitive to excessive downscaling. A 1024-pixel maximum side changed many outputs compared with the original images, while 1536 and 2048 were nearly or fully consistent with original-size behavior. The final baseline of 2048 is not chosen because it is always faster; it is chosen because it preserves current document behavior while bounding future oversized images.

Context tokens	Images/min	p95 (s)	Anchor coverage	Utility score
0	28.36	3.31	74.96%	88.05
2000	29.11	3.32	92.28%	96.46
5000	26.16	3.50	93.89%	93.22
10000	19.73	4.41	93.01%	82.92
20000	13.64	6.23	93.28%	73.15

Table 6.8: Captioning context-token sweep

The 2000-token setting gives much better grounding than zero context while avoiding the throughput and template-output penalties observed with long contexts.

The context-token sweep shows a trade-off: zero context is fast but less grounded, while very long context becomes slower and more template-like. The 2000-token setting gives much better anchor coverage than zero context while avoiding the throughput drop and over-contextualized output seen at 10000 and 20000 tokens. V2 therefore keeps 2000 as the inherited captioning quality baseline, but changes how the context is selected: it becomes section-aware rather than a generic neighboring-window context.

Concurrency	Images/min	p95 latency (s)	Decision
1	29.35	3.34	Single-thread baseline.
2	40.61	4.78	Large improvement.
4	59.67	5.97	Near throughput plateau.
6	68.18	7.60	Recommended v1 concurrency.
8	68.22	9.43	No throughput gain, worse tail latency.
10	67.39	10.68	Rejected.
12	67.17	12.06	Rejected.

Table 6.9: Captioning concurrency sweep

Concurrency 6 reaches the service plateau without the long-tail latency seen at higher values.

The final captioning recommendation combines quality and serving constraints. The setting `max_image_side=2048` preserves current-document outputs while protecting against future oversized images. The setting `max_context_tokens=2000` provides local semantic context for screenshots and diagrams. The throughput settings `concurrency=6`,

`batch_size=32`, and `checkpoint_interval=1` make the offline process repeatable. In v2, this baseline is kept, but the source of neighboring context is changed to the section-aware artifact.

Captioning choice	Selected value	Reason
Maximum image side	2048	Matches original-size outputs on the current corpus while bounding future large images.
Context budget	2000 tokens	Strong anchor coverage with much lower latency and less template output than long contexts.
Concurrency	6	Reaches the service throughput plateau without the long-tail latency of higher values.
Batch size	32	Balanced client scheduling and checkpoint recovery granularity.
Checkpoint interval	1	Negligible write overhead and best recovery safety.

Table 6.10: Captioning baseline inherited by the final system

V2 keeps these values and changes the context source to section-aware text.

6.5. First-Version Retrieval

The first retrieval experiments used a 220-question reviewed set and compared text and visual retrieval variants. The original direct text retrieval with `k=100` and `top_k=10` achieved any-gold MRR 0.8140, Recall@10 0.9955, and primary hit@10 0.95. Increasing `top_k` improved final gold coverage but expanded context substantially. The later primary-rank recalculation confirmed that large v1 chunks produced strong evidence coverage, especially for high-recall review settings.

The v1 retrieval stage was still primarily a retrieval-coverage study. It answered whether the enriched manual representation could recover reviewed evidence, not yet whether the final answerer would use that evidence under a strict multi-pass rubric. The 220 reviewed questions exposed recall and context-budget trade-offs at a larger scale than the later 50-question answering set. They also showed that visual retrieval could help some visual questions but was expensive and operationally fragile under the serving profile used at the time.

Configuration	MRR	Recall@10	Primary@10	Gold cov.@final	Avg context chars
Text `k=100 top=10`	0.8140	0.9955	0.9500	0.9406	31897
Text `k=100 top=20`	0.8140	0.9955	0.9500	0.9819	65774
Text `k=100 top=30`	0.8140	0.9955	0.9500	0.9883	99788
v1 `5000/500 high-recall`	0.7196	0.9545	0.9545	0.9973	181272
v1 `5000/500 interactive`	0.7172	0.9545	0.9545	0.9545	34468

Table 6.11: First-version retrieval results

The first version also included a route ablation that separated dense, sparse, and visual retrieval signals. This experiment explains why the later system keeps dense and sparse text retrieval enabled together. On the 220-question retrieval set, sparse-only retrieval was stronger than dense-only retrieval, but the fused dense+sparse text route was stronger than either single route. Visual-only retrieval could often identify relevant pages, but it was much weaker as a standalone chunk-selection mechanism.

Route	Primary MRR	Gold@10	Avg latency (s)	Avg context tokens
`dense+sparse+visual naive`	0.7622	0.9722	8.48	9090
`dense+sparse+visual bbox`	0.7287	0.9722	8.79	8492
`dense+sparse text`	0.7111	0.9586	0.31	8651
`sparse only`	0.6939	0.9242	0.02	9903
`dense only`	0.6392	0.9329	0.24	7666
`visual only naive`	0.5293	0.9505	6.19	8925
`visual only bbox`	0.3389	0.8744	6.31	6157

Table 6.12: First-version route ablation

The route ablation supports two design choices that remain visible in v2. First, dense and sparse retrieval have complementary value: dense retrieval helps with paraphrase and semantic proximity, while sparse retrieval is especially useful for exact menu labels, product terms, and short identifiers. Second, page-level visual retrieval improves the best retrieval-side ranking when it is added to dense+sparse text, but its latency is much higher and visual-only retrieval is not reliable enough to replace text. For this reason, v2 treats dense+sparse text retrieval as the fixed substrate and evaluates visual retrieval as an optional route rather than as the default. The ablation does not compare alternative dense embedding models or alternative sparse models; those remain a separate robustness question.

Two smaller v1 checks are also retained because they explain rejected implementation choices. First, seed expansion did not improve the 220-question candidate-generation

result over direct retrieval: both direct text and seed-expanded text reached the same 0.9545 primary-final success rate, but direct retrieval averaged 0.39 s while seed expansion averaged 1.80 s. Seed expansion increased online latency without a measured coverage gain, which is why it is not part of the final preset surface. Second, v1 tested the visual index construction settings used by the optional ColPali route. On 34 image/UI questions, visual-naive Primary MRR rose from 0.9020 at 150 dpi to 0.9160 at 200 dpi and then effectively plateaued at 250–300 dpi, while full-manual visual indexing time increased from 73.97 s at 150 dpi to 124.69 s at 300 dpi. The visual batch-size check showed that batch 8 was the fastest successful setting, 87.51 s, whereas batch 16 and 32 failed. These checks justify inheriting `RAG_FLOW_INDEX_VISUAL_DPI=200` and `RAG_FLOW_INDEX_VISUAL_BATCH_SIZE=8` for the optional visual index, while still leaving visual retrieval itself as a non-default route.

The v1 data showed strong evidence coverage, but also exposed the token cost of large top-k contexts.

The key lesson is that retrieval coverage and prompt usability diverge. The `top_k=30` text run reaches high gold coverage, but its average context is nearly 100k characters. The high-recall v1 chunk setting reaches even larger contexts. Such settings are useful as diagnostic upper bounds and manual-review modes, but they are not safe online defaults. V2 therefore evaluates retrieval through generated answers and context tokens rather than only through Recall@10 or MRR.

6.6. First-Version Answering

The v1 answering experiment used a 50-question answering set and an automatic coverage score. Text-only 16k obtained the best main score, 4.20, while text-only 10k was almost tied at 4.18 and used far fewer tokens. Adding rendered page images increased cost and latency without surpassing text-only.

The v1 answering set was intentionally visual-heavy for a first system test. It included image/UI/caption questions, visual text or operation questions, table-evidence questions, and nonvisual controls. It was designed to test whether manual RAG must handle evidence beyond plain paragraphs, not to estimate production traffic distribution. The automatic coverage score was less rigorous than the later v2 three-pass review, but it was sufficient to identify configurations that should not become defaults: page-image answering and thinking-on.

Run	Score	Usable	Visual score	Avg latency (s)	Avg tokens
Text 10k	4.18	0.78	4.11	5.50	10447.5
Text 16k	4.20	0.82	4.17	6.89	16415.8
10k + 200dpi x 3 pages	4.14	0.82	4.11	10.97	21880.3
16k + 200dpi x 3 pages	4.10	0.80	4.06	11.86	27786.4
16k + 250dpi x 3 pages	4.02	0.70	3.94	16.42	34131.5

Table 6.13: First-version answering results

Text-only 10k/16k were stronger defaults than adding rendered page images.

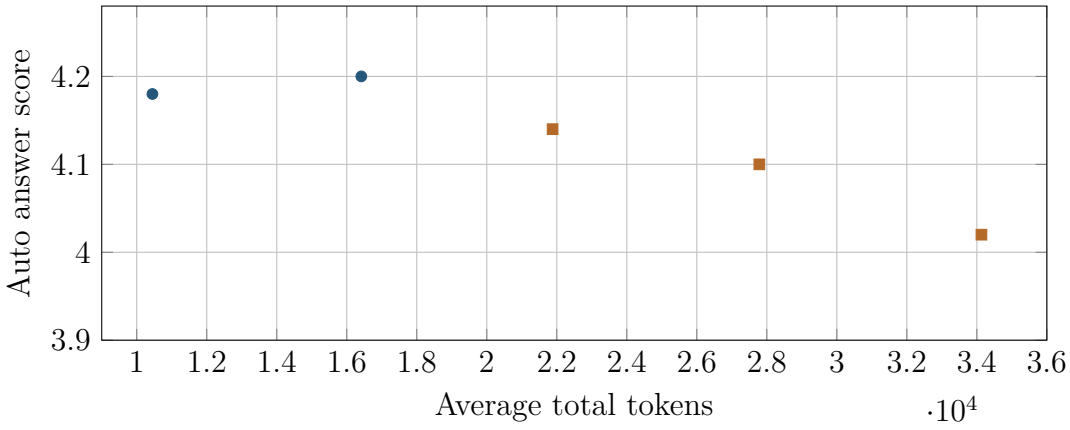


Figure 6.5: V1 answer score versus token cost

The v1 page-image results test the assumption that rendered PDF pages improve answers. In these runs, rendered page images increased total tokens and latency while failing to exceed text-only answering. The likely reason is that the images were full rendered pages rather than localized evidence crops. The model received more visual material, but not necessarily the right visual material in a form that was easy to use. V2 separates page-level visual retrieval, retrieval-provided image input, and text-only answering as distinct switches.

The v1 appendices also tested thinking and ColPali visual retrieval. Thinking-on reduced the score from 4.35 to 4.20 on the 20-question hard set and doubled latency. ColPali-on slightly improved the 36-question visual subset under a lower-memory serving profile, but retrieval latency increased from 0.39 s to 10.09 s in the documented comparison and default high-memory serving could not load it together with Qwen. These findings directly inform the final v2 presets: thinking remains off, ColPali becomes the optional `medium-with-visual-recall` mode, and page-image answering is not default.

Appendix run	Score	Usable	Coverage	Avg latency (s)	Tokens
Thinking off, hard 20Q	4.35	0.85	0.882	7.74	16400
Thinking on, hard 20Q	4.20	0.80	0.833	15.81	17400
ColPali off, visual 36Q, mem 0.70	4.06	0.69	0.822	7.29	16500
ColPali on, visual 36Q, mem 0.70	4.11	0.75	0.834	15.96	16600
ColPali off, visual 36Q, mem 0.75	4.17	0.78	0.839	7.95	16500

Table 6.14: First-version appendix checks for thinking and ColPali

The small quality movements are not sufficient to justify default thinking or default visual retrieval under the measured latency and memory constraints.

The thesis therefore does not treat multimodality as a single switch. V1 shows that multimodal preprocessing helps: patching and captioning improve the document representation before retrieval. The same experiments show that always-on multimodal answering is not automatically beneficial: page images and thinking add cost, latency, and failure risk. V2 keeps that distinction and tests it again with stricter scoring.

6.7. V1 Lessons Carried into V2

The first version leaves four concrete lessons for the final system.

1. **Preserve visual information during ingest.** Small icons and image descriptions are worth extracting before retrieval because they become searchable and auditable text evidence.
2. **Do not confuse throughput tuning with quality tuning.** DPI, image size, context budget, and chunk boundaries affect evidence quality; concurrency and batch size mainly affect serving behavior.
3. **Do not assume more context or more images are better.** V1 retrieval and answering show that large contexts and rendered pages can be expensive without improving answers.
4. **Keep visual retrieval optional.** ColPali can help recall visual pages, but the latency and serving constraints make it unsuitable as the default path without stronger evidence.

V2 tests these lessons under a stricter protocol. Chunking is retested because the v1 large-chunk baseline may be too noisy under final answer scoring. Retrieval is retested because

context, `k`, `top_k`, score-ratio pruning, and visual route interact with the new chunking profile. Answering is retested because the original image-input flag had to be audited and because final decisions require 200-question validation. The chapter sequence follows that refinement path: from system feasibility, to parameter evidence, to deployable presets.

7 | Second-Version Chunking and Indexing

7.1. Why Chunking Was Retested

The first version favored large, high-recall chunks around 5000 tokens with 500-token overlap. That was a sensible initial baseline because it maximized evidence coverage in a single-manual setting. The second version changes the problem. Source identity is now stable across multiple PDFs, sectioning is a formal upstream stage, image policies are carried as metadata, and final answers are scored directly rather than inferred from retrieval coverage. Under those conditions, chunking has to be retested as an answering-quality parameter, not only as a recall metric.

All v2 chunking experiments fixed the downstream conditions:

```
retrieval route text-only
retrieval_k 150
final_top_k 80
rrf_k 10
context cap 16000 retrieved-context tokens
score ratio 1.0
answering Qwen/SGLang, thinking off, no image input
QA set 50-question search set
```

Each profile regenerated the chunked JSON, rebuilt the text index, ran all 50 answers, and received three-pass Codex scoring.

Retesting chunking was necessary because v2 changed the meaning of a chunk. In v1, a chunk was mainly a large retrieval container for one long manual. In v2, a chunk is a source-aware evidence object with a normalized document path, section breadcrumb, page indices, block indices, optional image evidence, and a fixed Qdrant payload. These fields make smaller chunks more usable because evidence can be localized without losing

provenance. They also make stale indexes more dangerous because an old vector can point at a chunk id or source field that no longer matches the current artifact.

The v2 chunking experiment is both a quality test and a contract test. A profile must produce good answers, but it must also produce chunks that can be indexed, retrieved, displayed, and reviewed across 14 PDFs. The selected profile has to be stable enough to become the fixed upstream condition for all later retrieval and answering experiments, not merely the top row in one mean-score table.

7.2. Chunking Search Space

The v2 search evaluated 15 real answering runs:

- 8 coarse auto-mode profiles from 800 to 5000 max tokens.
- 4 local refinements around the best coarse point.
- 3 confirmation runs for minimum-token and mode effects.

The full single-parameter ablation grid was intentionally not run. The completed runs already gave a stable result: the best profile was a 1500-token section-aware chunk, and the large v1 chunks scored lower under final answer scoring. Each additional profile would require chunking, reindexing, 50 Qwen calls, and three Codex review passes. For that reason, the unexecuted full ablation is recorded as future robustness work rather than treated as completed evidence.

All rows are full answer-scored runs, not only retrieval hit-rate probes.

The original plan also described a full single-parameter ablation over many values of `max_tokens`, `overlap_tokens`, and `min_tokens`. That plan is kept in the archive as future robustness work, but it is not counted as completed evidence. The thesis claim is narrower and stronger: within the completed 15 full answer-scored runs, section-aware `1500/150/150` is the best supported final chunking profile, and the old `5000/500/200` profile is no longer the best choice under v2 answer-centered evaluation.

7.3. Chunk Construction Rules

The final chunker uses `auto` mode. When section metadata exists, blocks are grouped by `section_path`; within each section the chunker applies the token-window rules. When section metadata is absent, `auto` falls back to token-window mode over MinerU block order. This fallback is needed for short visual sheets with no outline, but it is not the

Stage	Completed profiles		Purpose
Coarse sweep	`800/100/100`, `1200/100/100`, `1500/150/150`, `2000/200/200`, `2500/250/200`, `3000/300/200`, `4000/400/200`, `5000/500/200`		Covers the main granularity range from small chunks to the v1 large-chunk baseline.
Local refinement	`1400/150/150`, `1600/150/150`, `1500/100/150`, `1500/200/150`		Checks whether the coarse optimum is caused by sparse sampling around the 1500-token point.
Confirmation	`1500/150/100`, `1500/150/200`, `1500/150/150`	`token`	Separates minimum-token sensitivity from the effect of section-aware `auto` mode.

Table 7.1: Completed v2 chunking search stages

common path for the long manuals.

The rules are designed for source-grounded retrieval, not only for vector indexing.

The overlap rule is intentionally modest. Overlap helps when a sentence, list, or table boundary falls near a chunk edge, but large overlap can make neighboring chunks nearly duplicate each other. In dense/sparse retrieval, duplicate chunks can crowd out distinct evidence in the final top-k. The selected overlap of 150 tokens is a continuity aid, not a substitute for a larger chunk.

Rule	Purpose
Do not cross section boundaries to fill a chunk	Avoid mixing unrelated procedures or UI flows merely to reach a token target.
Flush only after the soft minimum when a next item would exceed the maximum	Avoid very small chunks while preserving readable local units.
Use tail-token overlap rather than duplicating whole large blocks	Preserve local continuity without copying entire tables or long image descriptions.
Carry source and section metadata into every chunk	Allow retrieval outputs to be audited against original PDFs.
Carry image-answering evidence separately from text	Let image-input presets find images without making image input mandatory.

Table 7.2: Chunk construction rules used by the final v2 chunker

7.4. Results

Run	Mean	High	Low	Avg context	Avg total	Latency
`auto 1500/150/150`	2.5400	0.200	0.740	16717.36	17251.92	4.00
`auto 1500/150/200`	2.5000	0.200	0.740	16717.36	17257.02	4.04
`auto 2000/200/200`	2.4467	0.160	0.720	16594.40	17188.42	3.97
`auto 1400/150/150`	2.4400	0.200	0.720	16670.24	17212.58	4.02
`auto 800/100/100`	2.4400	0.240	0.740	16409.16	17341.22	4.09
`auto 1500/150/100`	2.4333	0.180	0.760	16723.34	17263.68	4.05
`auto 1600/150/150`	2.4067	0.140	0.740	16620.48	17196.58	3.99
`auto 2500/250/200`	2.4000	0.200	0.760	16625.48	17208.08	4.05
`auto 1500/200/150`	2.3933	0.180	0.760	16613.16	17211.94	4.04
`auto 1500/100/150`	2.3333	0.140	0.740	16595.72	17128.24	4.05
`auto 4000/400/200`	2.3333	0.160	0.780	16944.24	17575.74	4.20
`auto 1200/100/100`	2.3133	0.200	0.760	16492.78	17109.24	4.04
`token 1500/150/150`	2.2933	0.160	0.780	16859.96	17028.58	4.21
`auto 5000/500/200`	2.2867	0.160	0.780	16800.60	17494.76	4.01
`auto 3000/300/200`	2.2867	0.140	0.760	16672.52	17257.10	4.01

Table 7.3: All v2 chunking runs on the 50-question search set

Scores are three-pass Codex averages.

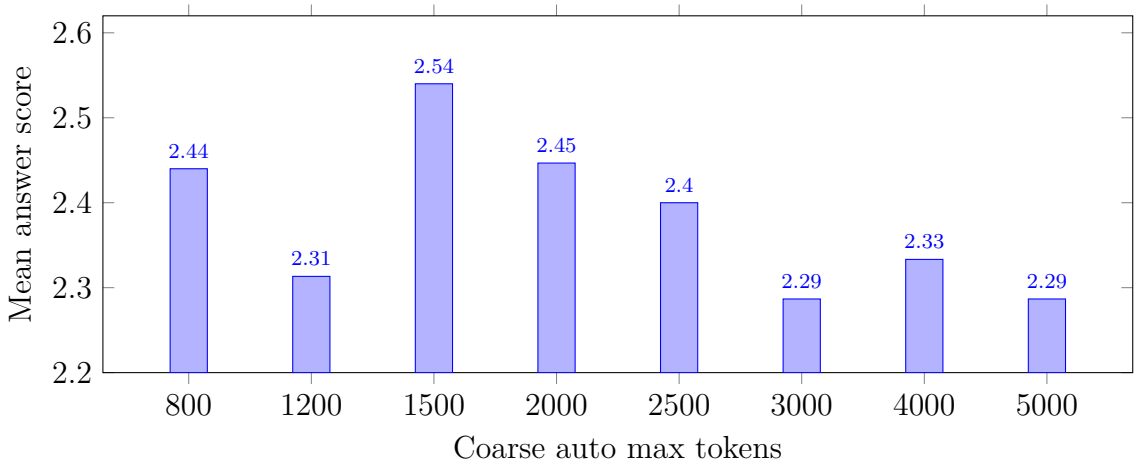


Figure 7.1: V2 coarse chunk-length search

The bar chart compares mean answer scores for different auto max-token settings and shows that section-aware chunks around 1500 tokens perform best, while larger chunks do not improve answers.

The ranking has two notable patterns. First, the best rows cluster around 1500–2000 tokens rather than at either extreme. Very small chunks can lose local conditions, while very large chunks add unrelated context. Second, `auto` mode matters. The pure token-window run with the same `1500/150/150` numbers scores much lower, which means the sectioning stage is doing more than adding display metadata. It changes the boundaries seen by retrieval and answering.

7.5. Interpretation

The best profile is:

```
RAG_FLOW_CHUNK_MODE=auto
RAG_FLOW_CHUNK_MAX_TOKENS=1500
RAG_FLOW_CHUNK_OVERLAP_TOKENS=150
RAG_FLOW_CHUNK_MIN_TOKENS=150
```

The large v1 profile, `5000/500/200`, drops to 2.2867. The pure token-window confirmation run also drops to 2.2933. These comparisons show that v2 section-aware boundaries are more than explanatory metadata; they improve answer quality by avoiding unrelated neighboring procedures, UI flows, and table fragments in the same chunk.

The same result explains why answer-centered scoring is necessary. A larger chunk can improve retrieval coverage by placing more text near the matched evidence, but the answerer still has to decide which sentence, row, or UI step matters. For technical manuals, long neighboring procedures often use similar terminology. A chunk about alarm configuration may sit next to unrelated device binding, map display, or record playback procedures. If all of this enters the same evidence block, dense retrieval can still look healthy while the final prompt becomes noisy. The v2 scoring detects that failure because it scores the final answer, not only whether the gold page or a supporting chunk was present.

Comparison	Better profile	Delta mean	Delta low-rate	Interpretation
`1500/150/150` vs `5000/500/200`	1500 profile	+0.2533	-0.040	Smaller section-aware chunks reduce prompt noise.
`1500/150/150` vs `token 1500/150/150`	auto mode	+0.2467	-0.040	Section boundaries add real answer value.
`1500/150/150` vs `1500/100/150`	150 overlap	+0.2067	0.000	Some overlap helps preserve local continuity.
`1500/150/150` vs `1500/150/200`	min 150	+0.0400	0.000	Minimum-size effect is small but does not beat the selected point.

Table 7.4: Chunking deltas against key alternatives

The selected profile is supported by two controls: the old v1 baseline and the pure token-window run.

Observed pattern	Evidence	Interpretation
Large chunks no longer win	`5000/500/200` scores 2.2867	High evidence coverage can be offset by prompt noise.
Pure token windows are weaker	Token `1500/150/150` scores 2.2933	Section boundaries add useful semantic structure.
Minimum-size effect is small	`1500/150/200` is close at 2.5000	The chosen `min=150` is stable but not a fragile magic number.
Overlap matters but is not monotonic	`1500/100/150` and `1500/200/150` are both below the selected row	Too little overlap loses continuity; too much can duplicate noise.

Table 7.5: Interpretation of the v2 chunking search

The selected profile is supported by patterns across the run family, not only by a single winning score.

7.6. Indexing Contract

The final indexing stage is not a new v2 parameter search. It fixes the Qdrant schema so chunking and retrieval can be evaluated cleanly. The final index used for retrieval and answering experiments contains 14 PDFs, the v2 chunking profile, and both text and visual indexes.

Item	Value	Meaning
Collection	`rag-flow-v2`	Dedicated experimental Qdrant collection.
Text vectors	`chunk-text-dense`, `chunk-text-sparse`	Chunk-level semantic and sparse retrieval.
Dense model	`intfloat/ multilingual-e5-large`	FastEmbed `TextEmbedding`; documents use `embed`, queries use `query_embed`.
Dense schema	1024-d cosine	Qdrant named vector; no additional dense-retrieval hyperparameters are tuned.
Sparse model	`Qdrant/bm25`	FastEmbed `SparseTextEmbedding`; exact-term and field-name recall.
Sparse schema	Qdrant IDF modifier	Stored as sparse indices/values; no custom BM25 `k1` or `b` is set.
Text batch size	256 chunks	Default batch size for dense and sparse text embeddings.
Text index time	612.184 s	Final v2 text-index write time.

Table 7.6: Final v2 text-indexing record

Item	Value	Meaning
Visual vectors	`page-image-colpali`	Page-level ColPali multivectors.
Visual model	`vidore/ colpali-v1.3-merged`	Fixed page-image retrieval model inherited from v1; not retuned in v2.
Visual schema	128-d multivector, MaxSim	Qdrant page-level multivector used only when visual recall is enabled.
Visual rendering	200 dpi, batch size 8	Inherited from v1 checks: 200 dpi reached the quality plateau and batch 8 was the fastest successful batch.
Visual index time	695.432 s	Final page-image index write time.

Table 7.7: Final v2 visual-indexing record

Retrieval experiments use the same fixed index.

The dense and sparse text indexes are fixed retrieval substrates in v2, not separate experi-

mental variables. Retrieval experiments vary candidate count, final top-k, RRF constant, context cap, score-ratio pruning, and visual route; they do not tune the dense encoder architecture or BM25 parameters.

Figure 7.2: Final v2 text-index and visual-index build times

The indexing times also show that index construction is an offline cost. Text indexing takes about 10 minutes and visual indexing about 12 minutes in the final record. These costs are acceptable during corpus ingestion but too expensive to repeat accidentally during retrieval experiments. The experiment scripts distinguish between changes that require rebuilding the index and changes that only affect query-time retrieval.

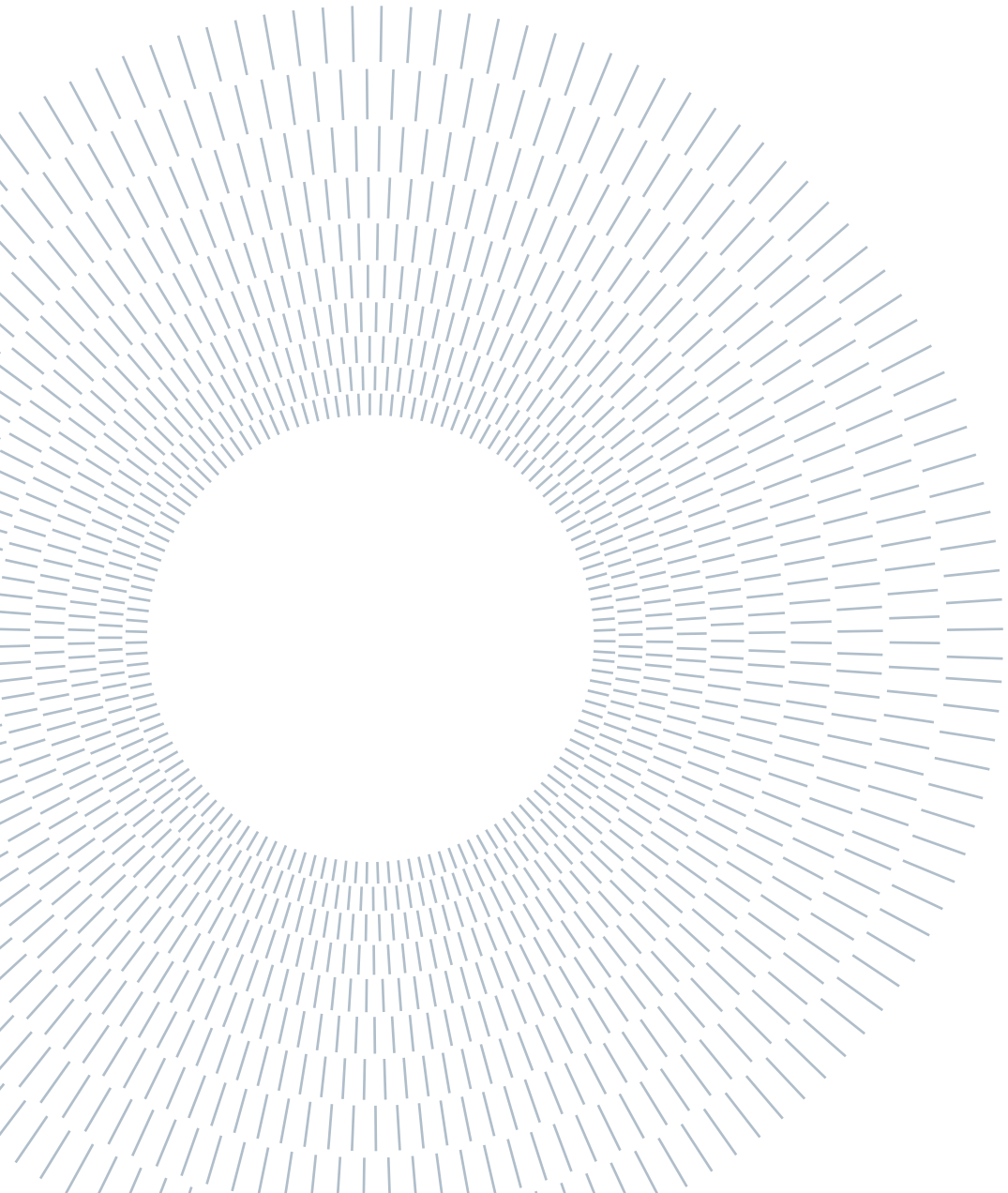
The key rule is that text and visual payloads must match the current chunked JSON. If chunk text, chunk ids, page indices, source metadata, bbox contribution, or image evidence changes, the affected index must be rebuilt. Retrieval and answering parameter changes do not require rebuilding the index.

The table records the rebuild boundary so that stale chunk payloads are not mixed with current retrieval and answering runs.

The rebuild boundary is one of the engineering contributions of the thesis. Without it, a retrieval experiment can accidentally combine a new configuration with stale Qdrant payloads. The result may still return text and may still produce fluent answers, but the scores would not correspond to the claimed configuration. Making the rebuild rule explicit keeps the experimental archive usable for final preset decisions.

Change	Required indexing action
Chunk text changes	Rebuild text index because dense and sparse vectors no longer represent the current chunk content.
Chunk id or block indices change	Rebuild text index and any visual payload that references chunk ids.
Source metadata or breadcrumb changes	Rebuild affected points so retrieval outputs and audit trails use current provenance.
Image evidence or bbox payload changes	Rebuild visual payloads before running visual route or image-input experiments.
Retrieval `k`, `top_k`, context cap, score ratio changes	No reindexing; these are query-time selection parameters.
Answering max tokens, thinking, image-input flag changes	No reindexing unless the image-input experiment also repairs stale image paths or payloads.

Table 7.8: Index rebuild boundary



8 | Retrieval Experiments

8.1. Task Definition

Retrieval starts after indexing and decides which evidence reaches the answering model. In v2, retrieval is evaluated by downstream answer quality. Each retrieval configuration produces a final output payload, the answering benchmark sends that payload to Qwen, and Codex scores the resulting answer. This avoids optimizing only for retrieval hit-rate while ignoring whether the answerer can use the evidence.

The tested variables include context cap, `retrieval_k`, `final_top_k`, RRF constant, score-ratio pruning, route mode, visual bonus, and visual weight. The original plan included local refinement around every numeric parameter. The actual run coverage was 37 retrieval-sweep runs plus 13 interaction runs. This is not a full Cartesian product, but it covers the regions needed for preset selection: small, medium, high, low-token, text-only, naive visual, and bbox visual modes.

Parameter family	Values explored	Decision use
Context cap	3k, 6k, 10k, 16k, 20k, 24k	Select online context budget and reject over-context modes.
Retrieval `k`	50, 80, 120, 150, 200	Separate candidate recall from final context size.
Final `top_k`	5, 10, 20, 30, 40, 60, 80	Choose count-level evidence cap for medium, low, and high modes.
RRF constant	5, 10, 30, 45, 60, 80, 100	Test dense/sparse fusion smoothness.
Score ratio	1.0, 0.8, 0.6, 0.4, 0.2, 0.0	Test relative tail pruning for compact contexts.
Route and visual weight	text-only, naive visual, bbox visual; weights 0.5, 1.0, 1.75, 2.5, 3.0	Decide whether ColPali changes the default or remains optional.
Interactions	13 representative full combinations	Check whether single-parameter winners remain good as deployable presets.

Table 8.1: Retrieval search coverage

The design is broad sequential search plus interaction checks, not a full Cartesian grid.

The reason for this search design is practical and methodological. A full Cartesian grid would multiply context caps, retrieval `k`, final `top_k`, RRF constants, score ratios, route

modes, visual weights, and answering calls. Many combinations are also uninformative: a visual weight without a visual route has no effect, and a 24k context cap under a 32768-token serving limit mainly tests failure behavior. The v2 method instead uses parameter-family sweeps to find plausible regions, then interaction runs and the final 200-question validation to test deployable combinations.

The retrieval chapter is a staged search over deployable behavior, not an attempt to discover an abstract optimum over every possible vector combination. It asks which evidence-selection strategy should be provided to users under a fixed chunking profile, fixed index, fixed answerer, and measured serving constraints. That is why some rows that score well in the 50-question search set still do not become defaults: a visual route may tie the best score but double latency, and a large context may look plausible on a small set but fail under final 200-question serving.

8.2. Retrieval Routes

The implementation supports three relevant routes:

```
text dense + sparse text retrieval
text-visual-naive text retrieval + ColPali page route + page broadcast bonus
text-visual-bbox text retrieval + ColPali page route + bbox-weighted bonus
```

The final answerer does not automatically receive images when visual route is enabled. Visual route changes retrieval ranking. Image input to Qwen is a separate answering-stage switch.

The route distinction affects the interpretation of the experiments. `text-visual-naive` means that ColPali finds visually relevant pages and the system broadcasts the page-level visual signal to chunks on those pages. It does not mean that the answerer sees the page image. `text-visual-bbox` tries to be more selective by using chunk/page geometry, but in the final experiments that fine-grained attribution is less stable than page broadcast. `RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=1` is a different switch: it appends image URLs from retrieved evidence to the answering payload. The switches are kept separate so that visual retrieval and multimodal answering can be judged independently.

Mode	What changes	What does not change
`text`	Dense and sparse text candidates are fused and truncated into a context.	No ColPali query; no images sent to Qwen by default.
`text-visual naive`	Page-level ColPali hits add a page-broadcast visual signal to chunk ranking.	Qwen still receives text only unless image input is separately enabled.
`text-visual bbox`	Page-level visual hits are mapped to chunks through bbox/page contribution.	Fine-grained attribution is heuristic and was not stable enough for final presets.
`FINAL_OUTPUT IMAGES=1`	Retrieval appends evidence image URLs to the final answer payload.	Text ranking can remain unchanged; image input is an answering-stage feature.

Table 8.2: Separation between retrieval route and image input

Keeping the switches separate prevents visual recall from being confused with multimodal answering.

8.3. Representative Results

Run	Mean	High	Low	Avg context	Avg total	Latency
`cap10 rk150 top20 naive w1`	2.5933	0.260	0.700	10341.84	10906.94	8.64
`cap10 rk80 top20 rrf10`	2.5933	0.240	0.720	10052.64	10510.52	3.32
`cap10 sweep`	2.5467	0.240	0.700	10559.98	11007.24	3.34
`top20 sweep`	2.5133	0.220	0.700	12271.92	12739.64	3.59
`top10 sweep`	2.5067	0.200	0.760	6068.08	6460.42	2.94
`cap10 rk80 top10 rrf10`	2.5000	0.180	0.720	6007.80	6392.16	2.94
`cap10 rk150 top10 ratio0.4`	2.4667	0.200	0.760	4211.22	4569.34	2.79
`naive w1 sweep`	2.4667	0.220	0.740	16437.52	17134.22	9.40
`rrf5 sweep`	2.4600	0.180	0.760	16629.80	17172.04	4.07
`bbox w3 sweep`	2.4533	0.220	0.740	16481.90	17430.20	9.34
`ratio0.4 sweep`	2.4400	0.200	0.740	5257.10	5620.24	2.90
`cap20k sweep`	2.3667	0.180	0.740	20673.02	21288.10	4.86
`cap24k sweep`	2.3533	0.160	0.760	24587.58	25267.90	5.21

Table 8.3: Representative v2 retrieval runs on the 50-question search set

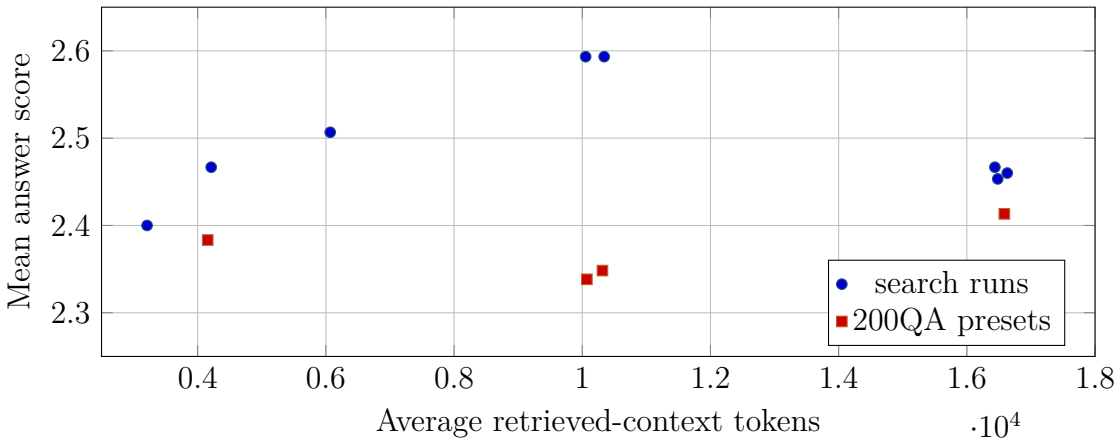


Figure 8.1: Trade-off between retrieved-context tokens and answer quality

The scatter plot uses average retrieved-context tokens on the x-axis and mean answer score on the y-axis; blue points are 50-question search runs and orange squares are 200-question final presets, allowing search-set preferences to be compared with final validation.

The representative results show why a single table sorted by mean score would be misleading. The best 50-question text-only interaction and the best naive visual interaction tie at 2.5933, but their latency differs substantially. The low-token interaction scores lower on the 50-question set but uses far fewer context tokens and later remains competitive

on 200 questions. The 16k and larger contexts do not dominate the search set, but the final `high` 16k preset obtains the best 200-question mean. These shifts justify using both search-set exploration and final-set confirmation.

8.4. Context Budget

The context-cap sweep shows that 10k is a strong search-set budget, not a universal optimum. In the 50-question sweep, 10k reaches 2.5467, better than 3k, 6k, 16k, 20k, or 24k in that stage. The later 200-question validation is flatter: 16k has the highest final mean score, while compact remains close with far fewer tokens. The final interpretation is a quality-cost plateau rather than a monotonic preference for more context.

The 24k setting is explicitly rejected. In final validation it creates 124 usage-missing cases out of 200 under the current 32768-token SGLang context limit. It is not a high-recall setting in practice; it is an over-context failure mode.

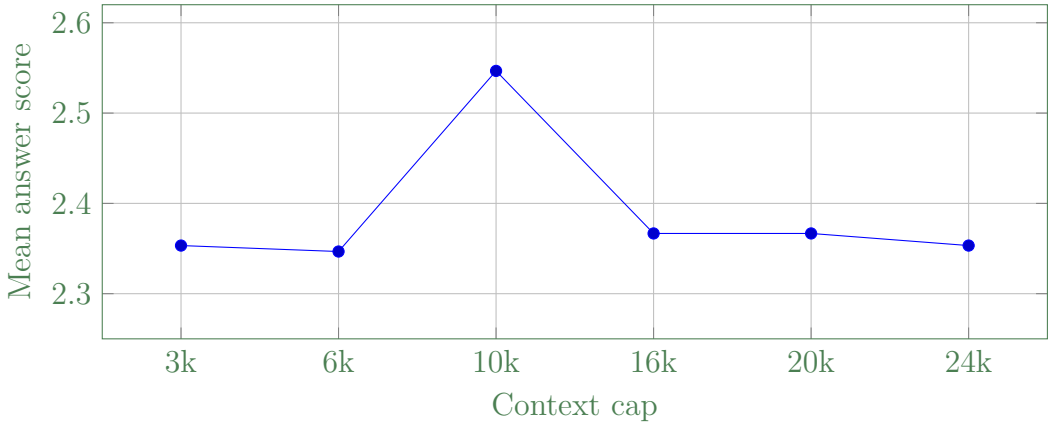


Figure 8.2: Context-cap sweep in the 50-question retrieval experiment

The line chart compares mean answer scores from 3k to 24k context caps, showing that 10k is strongest on the search set while 24k is later rejected because of serving context-limit failures.

Context cap	Mean	High	Low	Avg context	Efficiency
3k	2.3533	0.140	0.760	3542.32	0.6643
6k	2.3467	0.180	0.780	6535.64	0.3591
10k	2.5467	0.240	0.700	10559.98	0.2412
16k	2.3667	0.220	0.740	16604.66	0.1425
20k	2.3667	0.180	0.740	20673.02	0.1145
24k	2.3533	0.160	0.760	24587.58	0.0957

Table 8.4: Context-cap sweep on the 50-question retrieval search set

Small contexts are efficient but less stable; very large contexts have poor marginal value.

The context sweep has three separate conclusions. First, 10k remains a conservative medium-style budget because it performed well during search and keeps the implementation simple. Second, `low` is the better high-volume mode when token cost matters, because the final 200-question validation keeps comparable quality with much lower average context. Third, 16k is allowed as the `high` preset for difficult questions because it gives the highest final mean score without the operational collapse seen at 24k. The rejected 24k result is a reminder that a nominal context cap is not the same as a usable context cap. The answerer also needs room for prompt overhead, system instructions, and completion tokens.

8.5. Retrieval k and Final Top-k

The named `medium` preset chooses `k=80` and `top_k=20`. The search set has a tied best score between `cap10 rk150 top20 naive w1` and `cap10 rk80 top20 rrf10`; the text-only `k=80/top20` mode is retained as the conservative baseline because it avoids ColPali latency and keeps deployment simple.

The `low` mode deliberately uses `k=150/top10/score_ratio=0.4`. Keeping `k=150` preserves candidate recall, while `top_k=10` and score-ratio pruning remove low-value tail evidence. This is better than simply hard-capping context lower, because the retrieval pool can remain broad while the final context stays short.

Top-k sweep	Mean	High	Low	Avg context	Latency
`top5`	2.4000	0.160	0.800	3210.48	2.64
`top10`	2.5067	0.200	0.760	6068.08	2.94
`top20`	2.5133	0.220	0.700	12271.92	3.59
`top30`	2.3733	0.160	0.760	16075.14	4.03
`top40`	2.4600	0.200	0.760	16604.66	4.07
`top60`	2.4133	0.180	0.720	16604.66	4.02

Table 8.5: Final top-k sweep

Top-k controls count-level evidence breadth, but context cap and score pruning still determine actual prompt cost.

The top-k and retrieval-`k` sweeps also show that more candidates do not automatically mean better answers. Retrieval `k` controls how many candidates each retrieval route brings into the fusion stage; final `top_k` controls how many chunks can appear in the final context. A high retrieval `k` can help compact mode because score-ratio pruning still removes weak evidence after a broad candidate pool is considered. A high final `top_k`, however, can fill the prompt with redundant or neighboring chunks. The final presets use different combinations rather than a single global large-`k` rule.

Retrieval `k`	Mean	High	Low	Avg context	Latency
50	2.4067	0.180	0.740	16476.36	4.13
80	2.4267	0.180	0.740	16568.78	4.00
120	2.4067	0.160	0.760	16535.58	4.00
150	2.3667	0.220	0.740	16604.66	4.04
200	2.3600	0.160	0.800	16611.58	4.01

Table 8.6: Retrieval-k sweep under the 16k search condition

Increasing candidate recall beyond the useful range does not automatically improve generated answers.

8.6. RRF Constant

RRF controls how quickly lower-ranked candidates from each retriever lose influence. The isolated 16k sweep gives a mild advantage to `rrf_k=5`, but the interaction checks

show that `rrf_k=10` is the safer final constant. In the deployable 10k text interaction, `k=80/top20/rrf10` reaches 2.5933, while the corresponding `rrf5` interaction is only 2.4333. The final presets keep `rrf_k=10`: it is close to the best isolated sweep region, works in the strongest default interaction, and avoids overfitting the final choice to one 16k sweep row.

RRF setting	Mean	High	Low	Avg context	Latency
`rrf5 sweep`	2.4600	0.180	0.760	16629.80	4.07
`rrf10 cap16 baseline`	2.3667	0.220	0.740	16604.66	4.04
`rrf30 sweep`	2.4067	0.180	0.740	16586.34	4.05
`rrf45 sweep`	2.3933	0.200	0.740	16646.40	4.14
`rrf60 sweep`	2.3533	0.160	0.740	16648.82	4.15
`rrf80 sweep`	2.4267	0.200	0.740	16601.28	4.09
`rrf100 sweep`	2.3467	0.180	0.760	16685.02	4.10
`cap10 k80 top20 rrf10`	2.5933	0.240	0.720	10052.64	3.32
`cap10 k80 top20 rrf5`	2.4333	0.200	0.760	10029.42	3.31

Table 8.7: RRF sweep and interaction check

The final RRF constant of 10 is based on deployable interaction behavior, not only the isolated 16k sweep.

The RRF result is a case where interaction checks change the choice. In isolation, a smaller RRF constant can look attractive because it emphasizes the very top hits. In a deployable 10k text setting, however, `rrf_k=10` performs better than `rrf_k=5`. The final choice is not the best number from a single sweep; it is the number that behaves well under the final medium-style interaction.

8.7. Score Ratio

`RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO` uses allowed-drop semantics: `1.0` means no relative pruning; `0.4` keeps candidates close enough to the best score while dropping weak tail evidence. The final `low` preset uses `score_ratio=0.4`. On the 50-question interaction run it reaches 2.4667 with 4211.22 average context tokens. In the 200-question validation it becomes the most efficient preset: mean score 2.3833 with 4156.10 average context tokens.

Score ratio	Mean	High	Low	Avg context	Efficiency
0.0	2.2667	0.120	0.780	751.62	3.0157
0.2	2.2867	0.140	0.800	2208.04	1.0356
0.4	2.4400	0.200	0.740	5257.10	0.4641
0.6	2.4133	0.200	0.780	11574.16	0.2085
0.8	2.3800	0.180	0.760	16479.64	0.1444
1.0	2.4333	0.180	0.700	10034.18	0.2425

Table 8.8: Score-ratio sweep

Very aggressive pruning is efficient but loses evidence; 0.4 becomes the useful low-token pruning point.

The ratio result is also a caution. The most efficient row, `0.0`, is not a final preset because it keeps too little evidence and lowers answer quality. The final `low` preset is not the smallest possible prompt; it is the smallest prompt that remains competitive in the final 200-question validation.

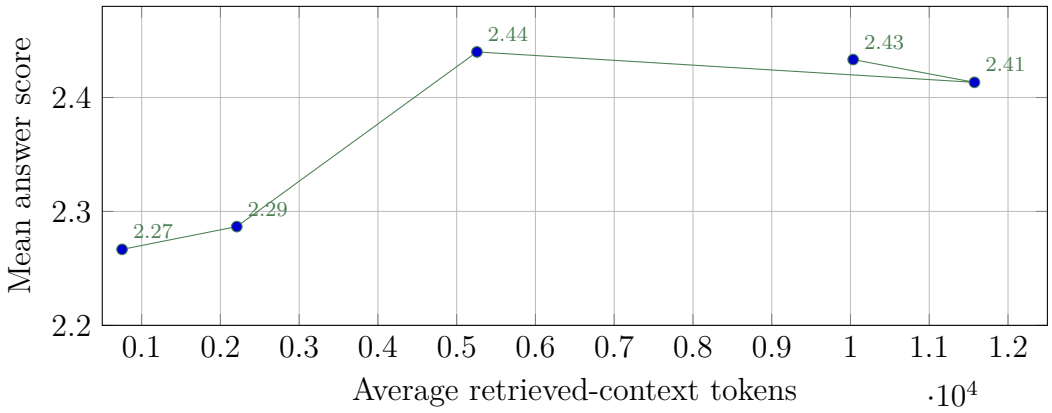


Figure 8.3: Score-ratio pruning trade-off between quality and context cost

The scatter plot compares average context tokens and mean answer score for different score-ratio settings, showing that very aggressive pruning saves tokens but loses too much answer quality.

8.8. Visual Retrieval

Naive visual route is the only visual retrieval mode retained as a supported preset. In the 50-question interaction run it ties the highest mean score, 2.5933, but latency rises

to 8.64 s. In the 200-question validation, `medium-with-visual-recall` obtains 2.3483 mean score with 8.43 s latency, close to medium text but far slower. BBox attribution is worse in the final validation, with mean score 2.1900. The likely reason is that page-level ColPali signals are reliable for recalling a page but not yet reliable enough to assign fine-grained chunk geometry without calibrated patch attribution.

Visual configuration	Mean	High	Low	Avg context	Latency
`text-only default interaction`	2.5933	0.240	0.720	10052.64	3.32
`naive w=1 interaction`	2.5933	0.260	0.700	10341.84	8.64
`bbox w=3 interaction`	2.2667	0.140	0.800	9727.80	8.51
`naive w=1 200QA`	2.3483	0.200	0.745	10314.86	8.43
`bbox w=3 200QA`	2.1900	0.155	0.775	9599.53	8.27

Table 8.9: Visual route comparison

Naive visual retrieval remains a recall mode, while bbox visual attribution is not reliable enough for final presets.

The visual result is not a dismissal of visual retrieval. It shows that the current page-level visual route can act as a recall option but is not sufficient to replace text as the default. Many visual questions in the benchmark are already partially handled by patching and captioning, so the marginal gain from page-level ColPali is smaller than one might expect. At the same time, visual route latency is consistently much higher because the system must run page-image retrieval and merge those results with text candidates. The final preset `medium-with-visual-recall` is provided for cases where the user or application expects a visual page to matter.

8.9. V3 Text-Retrieval Backbone Check

After the final v2 preset strategy was established, a third experiment checked whether the remaining weak RAG answer quality was mainly caused by the text-retrieval backbone. This experiment fixed the source PDFs, sectioned/patched/captioned/chunked JSON, section-aware `1500/150/150` chunks, answer prompt, Qwen answerer, text-only route, no image input, no visual retrieval, and thinking off. Only the dense model, dense vector size, sparse model, dense/sparse fusion, and RRF constant were changed.

The scoring protocol was intentionally narrow. For each candidate answer, the Codex reviewer received only the question, canonical answer, and candidate answer. It did not

receive retrieved context, gold evidence, PDF pages, source paths, previous scores, or retrieval metadata. The reviewer model was Codex `gpt-5.5` with `xhigh` (extra-high) reasoning effort. For high-memory embedding runs, retrieval and answer generation were split into two stages: the exact final context was first archived for every question, and Qwen later generated answers from those archived contexts. The reported total latency is retrieval latency plus answer-generation latency, so it still represents the complete question-to-answer path.

Single-route row	Mean	High	Low	Avg ctx	Latency	Index/DB
`dense qwen3-4b`	1.4800	0.280	0.720	7495.08	3.64	1004.74s / 36.49MB
`sparse bm25`	1.3667	0.240	0.740	10244.32	4.42	4.26s / 20.33MB
`sparse bge-m3`	1.3467	0.240	0.740	10528.84	4.47	33.88s / 21.60MB
`dense qwen3-0.6b`	1.3133	0.260	0.740	7554.20	3.70	39.97s / 18.94MB

Table 8.10: V3 single-route finalists under content-only Codex scoring

The best dense-only route is `Qwen/Qwen3-Embedding-4B` with a 2560-dimensional dense vector. The best sparse-only route remains the existing `Qdrant/bm25` route. `BAAI/bge-m3` is close, but it does not beat BM25 under the three-pass finalist rescore.

The full Phase 1 screening table is kept in table A.7. It includes every planned dense and sparse candidate, including the 4096-dimensional `Qwen/Qwen3-Embedding-8B` row. The main text reports only the four finalists after three-pass rescore, so small differences from the appendix screening means are expected.

Fusion/RRF row	Mean	High	Low	Avg ctx	Latency	RRF
`current + bm25`	1.3133	0.220	0.760	10300.66	4.85	10
`qwen3-4b + bm25`	1.3333	0.220	0.760	9908.98	4.33	10
`qwen3-4b + bm25`	1.3867	0.240	0.740	9902.66	4.25	5
`qwen3-4b + bm25`	1.3800	0.240	0.720	9709.80	4.34	60

Table 8.11: V3 fused-route and RRF screening on 50 questions

The fused improvement on the 50-question set is positive but small. With the same `rrf_k=10`, replacing only the dense route increases the mean score from 1.3133 to 1.3333. A small RRF sweep then selects `rrf_k=5`, which reaches 1.3867 on the screening set.

200Q row	Mean	High	Low	Avg ctx	Latency	Avg tokens	Index/DB
`qwen3-4b + bm25 + rrf5`	1.3250	0.235	0.745	9906.70	4.44	10443.75	93.59s / 38.09MB
`current + bm25 + rrf10`	1.2633	0.205	0.760	10251.18	4.95	10742.45	604.44s / 20.33MB

Table 8.12: V3 200-question confirmation for the selected text backbone

On the final 200-question confirmation, the selected v3 text backbone improves the mean score by 0.0617, raises the high-quality rate by 0.030, and lowers the low-score rate by 0.015. It also uses slightly fewer retrieved-context tokens and Qwen tokens in this run. The larger dense model increases index dimensionality and collection size, so the result should be interpreted as a quality/cost trade-off rather than a cost-neutral improvement.

The judgment is conservative. Text-backbone choice matters: the best tested v3 row uses Qwen3-Embedding-4B dense retrieval, BM25 sparse retrieval, and `rrf_k=5`. However, the gain is modest and does not close the broader direct-PDF diagnostic gap. This means the remaining weakness is not explained only by the dense model, sparse model, or RRF constant. The evidence points instead to a combined bottleneck: PDF evidence preservation, fragmented diagrams, caption limits, chunk/evidence packaging, retrieval-to-answer selection, and answerer capability.

Because the selected v3 row used a medium-style retrieval budget, a final matched-control experiment was added against the original v2 `high` preset. This control used the same 200 questions and fixed the high-recall parameters: text-only retrieval, BM25 sparse retrieval with the current IDF sparse-vector policy, section-aware `1500/150/150` chunks, `k=150`, `top_k=80`, 16k context cap, `rrf_k=10`, no visual retrieval, no image input, and thinking off. The only retrieval-backbone change was replacing `intfloat/multilingual-e5-large` with `Qwen/Qwen3-Embedding-4B`. The comparison rows were then rescored together with the same content-only Codex `gpt-5.5/xhigh` reviewer, three passes, using only question, canonical answer, and candidate answer.

The matched-control rescore is a separate content-only batch from the earlier direct-PDF diagnostic in table 9.6. The direct-PDF answer is kept as the same source-PDF reference condition, but the value reported below belongs to the five-row matched-control batch and is compared with the rows in table 8.13.

Matched-control row	Answerer	Mean	High	Low	Avg ctx	Q-to-A (s)
`v2 high baseline`	Qwen	1.3567	0.225	0.735	16584.94	4.17
`v2 high + Qwen3-4B`	Qwen	1.3200	0.210	0.755	16577.42	9.76
`Codex over v2 high ctx`	Codex	1.6842	0.295	0.655	16584.94	15.02
`Codex over Qwen3 ctx`	Codex	1.7250	0.305	0.675	16577.42	13.58
`Codex direct PDF`	Codex	2.7617	0.380	0.490	n/a	46.81

Table 8.13: Additional matched v2-high control using the same 200 questions

table 8.13 shows that dense-backbone replacement alone does not improve the deployed

Qwen answer path under the original high-recall budget. The Qwen row drops from 1.3567 to 1.3200 and becomes slower. The same new context helps Codex slightly, from 1.6842 to 1.7250, suggesting a small evidence-exposure improvement for a stronger answerer. The direct-PDF row remains much higher. The earlier v3 gain is therefore treated as a modest medium-budget configuration improvement, not as evidence that Qwen3-4B dense retrieval fixes the v2 high preset by itself.

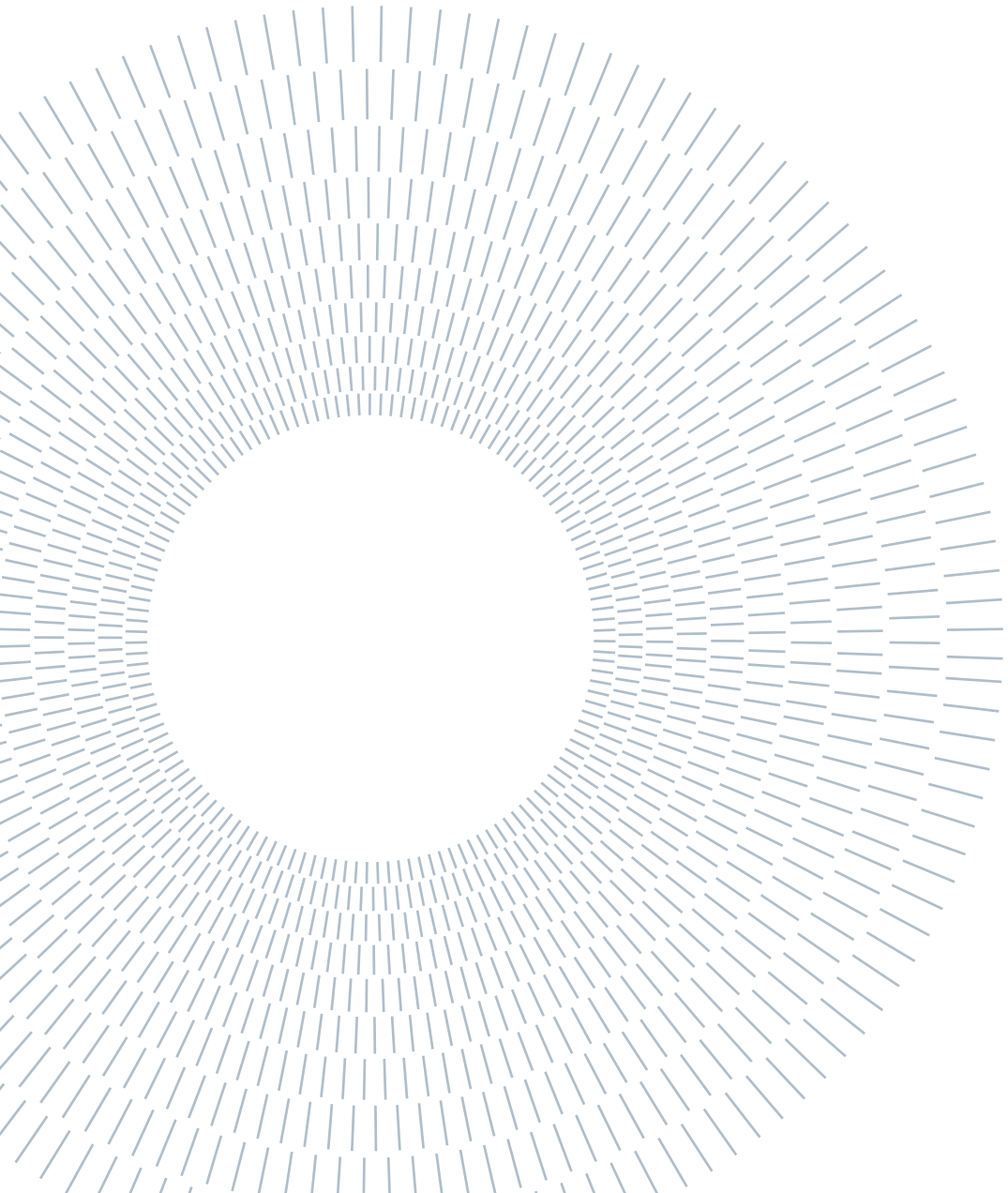
8.10. Final Retrieval Choices

The retrieval-stage conclusions are:

- `default/medium`: text-only, `k=80`, `top_k=20`, 10k cap, `score_ratio=1.0`.
- `high-recall/high`: text-only, `k=150`, `top_k=80`, 16k cap.
- `compact/low`: text-only, `k=150`, `top_k=10`, 10k cap, `score_ratio=0.4`.
- `visual-recall/medium-with-visual-recall`: `text-visual-naive`, page-naive visual bonus, `visual_weight=1.0`, `k=150`, `top_k=20`, 10k cap.
- `text-visual-bbox`, 24k context, and seed expansion are not recommended defaults.

The v3 backbone check does not change the preset names. It refines the text-backbone option behind the medium-style text route: when embedding resources permit, Qwen3-Embedding-4B with BM25 and `rrf_k=5` is the stronger tested medium-budget text-backbone candidate. The matched high-control does not support replacing the v2 `high` dense backbone by Qwen3-4B alone. The original e5-large with BM25 and `rrf_k=10` remains the lower-resource audited high-recall baseline, and v3 does not retune the low, high, or visual presets.

The retrieval choices become the backbone of the final preset strategy. The historical `default` row maps to `medium`, a conservative general-purpose baseline. `high` accepts more context for difficult questions, while `low` uses a broad candidate pool followed by stricter final selection. It is the preferred high-volume mode when token cost or throughput dominates. `medium-with-visual-recall` adds page-level visual evidence to ranking but keeps the answerer text-only. The rejected options are also part of the final design: bbox visual attribution, 24k context, and always-on visual routing would make the system more complex without a stable quality gain under the measured conditions.



9 | Answering Experiments and Final Presets

9.1. Answering Boundary

Answering consumes the retrieval final output. It does not choose pages, rerank chunks, or render PDFs itself. The answering-stage variables are:

- output token budget,
- thinking on/off,
- whether retrieval-provided evidence images are appended as image input.

The final answerer uses Qwen through an SGLang OpenAI-compatible service. The fixed default answer prompt is evidence-bound: the answer must rely on retrieved context and should not invent unsupported facts.

The boundary is necessary because otherwise an answering experiment can silently become a retrieval experiment. If the answerer were allowed to fetch extra pages, rerender PDFs, or choose its own visual evidence, then a change in answer quality could not be attributed to the configured retrieval payload. RAG-Flow deliberately freezes the evidence at the retrieval final output. The answerer receives that payload, generates an answer, and the review protocol judges whether the payload was sufficient and whether the answerer used it correctly.

9.2. Experimental Coverage

The answering experiments are deliberately narrower than retrieval experiments because answering is the most expensive stage: each configuration requires Qwen generation and three separate Codex review passes. The completed answering coverage still tests every final decision that changes the answer payload.

Question	Completed coverage	Validation level	Final consequence
How large should the answer budget be?	`max_tokens` sweep: 1000, 2000, 4000, 6000, 8000	5 runs x 50 questions	Keep 8000 as truncation insurance.
Should thinking be enabled?	Thinking off/on on the 50-question feature set and again on 200 questions	50Q plus 200Q	Reject thinking as a default and as a final preset.
Does image input help?	Historical image-flag audit, repaired 50Q image run, repaired 200Q image run	50Q plus 200Q with verified image URLs	Keep image input as diagnostic, not default.
Which complete modes are deployable?	11 final 200-question runs	2200 generated answers and 6600 review judgments	Define six runnable modes, mark measured versus derived rows, and keep historical names as compatibility aliases.

Table 9.1: Answering-stage experimental coverage

The stage changes only generation budget, thinking, and retrieval-provided image input; retrieval route and context selection are evaluated in the previous chapter.

The coverage table also explains why the thesis does not run every answering switch against every retrieval switch. The main retrieval decisions have already been made in the previous chapter. The answering chapter asks a narrower question: once retrieval has selected evidence, should the answerer receive more output budget, hidden thinking, or images? This keeps the causal interpretation clean. A bad image-input result should not be blamed on a different retrieval route, and a thinking-on result should not be mixed with an unrelated context-cap change.

9.3. Output Token Budget

The 50-question sweep tested `max_tokens=1000,2000,4000,6000,8000`. The highest score is at 8000, but 6000 and 8000 have nearly identical usage because typical answers do not reach the cap. The final preset keeps 8000 as a safety margin against truncation, not as an instruction to write longer answers.

Run	Mean	High	Low	Avg total tokens	Latency
`max1000`	2.5000	0.180	0.700	10510.52	3.31
`max2000`	2.4933	0.180	0.720	10510.52	3.32
`max4000`	2.4733	0.240	0.720	10510.52	3.34
`max6000`	2.5533	0.220	0.700	10510.52	3.32
`max8000`	2.5667	0.220	0.700	10510.52	3.36

Table 9.2: Answering output token sweep on the 50-question set

The planned local four-point refinement around the best value was intentionally not executed. The coarse sweep already showed the operational fact that all five rows have the same observed average total-token usage, 10510.52, and nearly identical latency. That means the model did not normally reach even the lower output limits in this task. A local sweep around 8000 would mostly measure scoring noise rather than a true budget effect. Keeping 8000 is a conservative engineering choice because it protects long procedural answers from truncation without increasing observed average usage.

The 8000-token value is a ceiling, not a target. The prompt remains evidence-bound and asks for the necessary answer, not for a long essay. The ceiling exists because some manual questions require ordered steps, prerequisites, parameter values, and warnings. A smaller ceiling can truncate those rare long answers, while the observed usage records show that the larger ceiling does not inflate average usage when the model has enough evidence to answer concisely.

9.4. Thinking and Image Input

Thinking is not selected for the final presets. On the 50-question answering feature run, thinking-on scores 2.3200 and raises latency to 12.64 s, compared with 2.5600 and roughly 3.33 s for text-only thinking-off. The final 200-question run confirms the negative trend: thinking-on scores 2.0233 with 13.30 s latency.

The negative thinking result is plausible for this task. Many failures are not abstract reasoning failures; they are evidence-selection and grounding failures. If the necessary row, screenshot label, or procedure step is missing from context, hidden reasoning cannot recover it. If the evidence is present but surrounded by many similar procedures, extra reasoning can still choose the wrong neighbor. In the measured runs, thinking increases latency and severe failures rather than resolving the main bottleneck.

Image input is more subtle. An early image flag run did not actually deliver image URLs, and is retained only as an invalid historical audit row. After fixing image paths and rebuilding the index, real image input is confirmed: the 50-question fixed run sends 359 image URLs, and the 200-question fixed run sends 1378 image URLs. However, quality does not improve.

Run	Mean	High	Low	Avg total	Latency	Image URLs
Text fixed index, 50Q	2.4200	0.160	0.760	10722.22	3.31	0
Real images fixed, 50Q	2.3267	0.220	0.740	14160.38	4.74	359
Default text, 200Q	2.3383	0.190	0.750	10588.75	3.37	0
Real images fixed, 200Q	2.3183	0.190	0.740	14325.33	4.65	1378

Table 9.3: Real image input reaches the answerer but does not improve mean score

The repaired image-input result separates two possible explanations. Before the repair, a good score under `FINAL_OUTPUT_IMAGES=1` could be mistaken for evidence that images help. The audit showed that the image count was zero, so the run was actually a text run with a misleading flag. After the repair, real image URLs reached the answerer, LLM errors remained absent, and no thinking leaks were recorded. The score still decreased. The conclusion is based on the repaired image-input run rather than on the invalid zero-image audit row.

The repaired runs also clarify the future direction. The limitation is payload granularity rather than image evidence in general. Retrieval can attach many full evidence images, including screenshots or pages whose useful region is only a small part of the image. The answerer then pays image-token cost and must localize the relevant region itself. A better future design would pass cropped, bbox-localized evidence images or a much smaller image set, ideally selected by calibrated visual attribution.

Image audit row	Questions	Image URLs	Mean	Avg total	Latency	Interpretation
Historical 200Q image flag	200	0	2.3833	10588.75	3.37	Invalid as image input; retained only to document the zero-image path.
Repaired 50Q image input	50	359	2.3267	14160.38	4.74	Valid image input; quality and token efficiency decrease.
Repaired 200Q image input	200	1378	2.3183	14325.33	4.65	Valid final check; images are diagnostic, not default.

Table 9.4: Image-input audit

The invalid historical flag is kept separate from the repaired runs that actually transmitted image URLs.

9.5. Final 200-Question Validation

The final validation compares representative complete configurations. It includes default text, high recall, compact, a small-context diagnostic configuration, visual recall, bbox

visual, thinking-on, real image input, and the 24k over-context stress case.

Configuration	Mean	High	Low	Avg context	Avg total	Efficiency	Latency
`high 16k`	2.4133	0.170	0.740	16584.94	17198.90	0.1455	4.17
`compact ratio0.4`	2.3833	0.170	0.770	4156.10	4565.19	0.5735	2.79
`historical image audit`	2.3833	0.175	0.755	10071.58	10588.75	0.2366	3.37
`3k top5 diagnostic`	2.3550	0.150	0.775	2687.77	3024.95	0.8762	2.58
`visual naive w1`	2.3483	0.200	0.745	10314.86	10965.50	0.2277	8.43
`default text`	2.3383	0.190	0.750	10071.58	10588.75	0.2322	3.37
`default real images`	2.3183	0.190	0.740	10251.18	14325.33	0.2262	4.65
`compact top10`	2.2967	0.165	0.765	5823.84	6274.21	0.3944	2.92
`visual bbox w3`	2.1900	0.155	0.775	9599.53	10413.93	0.2281	8.27
`thinking on`	2.0233	0.175	0.730	10071.58	11930.10	0.2009	13.30
`very high 24k`	0.8083	0.055	0.900	24531.97	9293.99	0.0330	2.73

Table 9.5: All final 200-question validation runs

The historical image audit row is retained to document the pre-repair zero-image path; it is not treated as an image-input result.

The final table is a deployment comparison, not only a score ranking. The highest score is `high 16k`, but its average context is about four times that of `low`. The smallest deployable context is `low ratio0.4`, not the 3k diagnostic, because the 3k diagnostic has lower quality and is not provided as a preset. The visual and image-input rows are retained because they test concrete design choices, even though they do not win the default decision. The 24k row documents the serving boundary and prevents an unsafe high-context preset from becoming a supported mode.

9.6. Direct-PDF Diagnostic Baseline

After the final preset validation, an additional diagnostic baseline was run to separate two questions that the normal RAG experiment combines: whether the source PDFs contain enough information to answer the benchmark, and how much information is lost by the current parsing, chunking, retrieval, and evidence-selection pipeline. In this baseline, Codex was given the question and the original source-PDF paths for that question, but not RAG-Flow chunks, Qdrant results, generated captions, evidence cards, gold answers, thesis text, or web search. Codex then inspected the original PDFs directly and produced a candidate answer.

Five deployed preset outputs and the direct-PDF baseline were rescored with the same content-only reviewer prompt. The reviewer received only the question, the canonical

answer, and the candidate answer. It did not receive retrieved context, gold evidence, source-PDF paths, or retrieval metadata. As summarized in table 5.8, table 9.6 is a separate diagnostic rescore and is kept separate from the RAG-evidence scoring used in the main final 200-question validation.

Row	Mean	Median	High-quality rate	Low-score rate	Avg retrieved context
`low`	1.1933	0.0	0.190	0.755	4156.10
`medium`	1.3250	0.0	0.215	0.740	10071.58
`high`	1.3383	0.0	0.225	0.730	16584.94
`medium-with-image-input`	1.3950	0.0	0.220	0.720	10251.18
`medium-with-visual-recall`	1.3842	0.0	0.225	0.725	10314.86
`Codex direct PDF`	2.8383	2.5	0.400	0.475	n/a

Table 9.6: Content-only diagnostic rescore of selected presets and a Codex direct-PDF baseline

The direct-PDF row receives a higher score than the RAG preset rows under this content-only rescore. The comparison includes two differences at once: a different answerer and a different evidence-access procedure. The direct-PDF baseline used the Codex CLI default configuration on this machine, `gpt-5.5` with `xhigh` reasoning effort, and it allowed an agentic model to navigate the original PDF files directly. Its advantage may partly reflect a stronger underlying model and stronger document-navigation behavior.

To separate model strength from RAG evidence loss, a second diagnostic run reused the archived `high` preset context rather than the original PDFs. Codex received the question and the exact saved RAG context file for that question, but it was not allowed to inspect source PDFs, Qdrant indexes, previous answers, gold answers, thesis text, or the web. The resulting answer was then scored by the same content-only reviewer. This creates a middle row: the answerer is the same Codex setup as the direct-PDF baseline, but the evidence is frozen to what RAG-Flow exposed.

Row	Mean	Median	High-quality rate	Low-score rate	Zero-score rate
`high RAG answer`	1.3383	0.0000	0.225	0.730	0.640
`Codex from high RAG context`	1.7367	0.1666	0.295	0.665	0.500
`Codex direct PDF`	2.8383	2.5000	0.400	0.475	0.000

Table 9.7: Content-only diagnostic comparison between high RAG context and direct PDF access

table 9.7 separates two effects. First, Codex improves over the original `high` RAG answerer

when it is given the same archived RAG context: mean score rises from 1.3383 to 1.7367, with 53 improved questions, 132 unchanged questions, and 15 worse questions. Answerer capability therefore matters. Second, direct PDF access remains much stronger than Codex from archived RAG context: mean score rises from 1.7367 to 2.8383, with 127 improved questions, 63 unchanged questions, and 10 worse questions. The remaining gap suggests that the current RAG context often does not expose enough of the original document information to the answerer, even when the answerer is stronger. The likely causes are implementation-level evidence losses during PDF parsing, visual preservation, chunking, retrieval, or answer-evidence selection. The direct-PDF row is used as a source-PDF reference point for improving the pipeline.

The diagnostic baselines have different operating costs, so their timing is stated by procedure. table 9.8 reports a comparable question-to-answer elapsed time for each row. For the normal `high` RAG answerer, this is the recorded online RAG total from question to answer: retrieval, context assembly, and Qwen generation. For `Codex from high RAG context`, the generation run reused archived contexts; to make the number comparable, the table adds back the source `high` run's retrieval latency to Codex's answer elapsed time. For `Codex direct PDF`, the elapsed time is the direct agentic procedure from question and local PDF paths to answer, with no RAG retrieval stage. The total time is a serial-equivalent sum of per-question elapsed time; actual wall-clock time can be lower when the runner uses concurrency. Token accounting is fully available for the normal RAG answerer because the Qwen service reports usage. Codex CLI did not provide decomposable prompt/completion token usage for the diagnostic rows, so the table reports the fixed archived-context estimate only where it can be computed from the saved RAG context. The direct-PDF row has no corresponding fixed RAG context and is reported as N/A for token cost.

Row	Avg Q-to-A (s)	Serial total (min)	Max Q-to-A (s)	Procedure
`high RAG answer`	4.17	13.89	13.00	RAG + Qwen
`Codex from high RAG context`	15.02	50.06	38.42	RAG + Codex
`Codex direct PDF`	46.81	156.04	223.37	Direct PDF agent

Table 9.8: Operational cost of the context-only and direct-PDF diagnostic baselines

Note. Q-to-A means question-to-answer elapsed time. The first row is the measured online RAG path with retrieval, context assembly, and Qwen generation; its usage is 3.44M measured LLM tokens, with 3.32M estimated retrieved-context tokens. The second row adds the source `high` retrieval latency to the Codex answer time over the archived `high`

context, whose fixed context estimate is also 3.32M tokens. Codex CLI did not provide decomposable prompt/completion token usage for the diagnostic rows. The table reports only the fixed archived-context estimate where available; the direct-PDF row has no fixed RAG context to estimate.

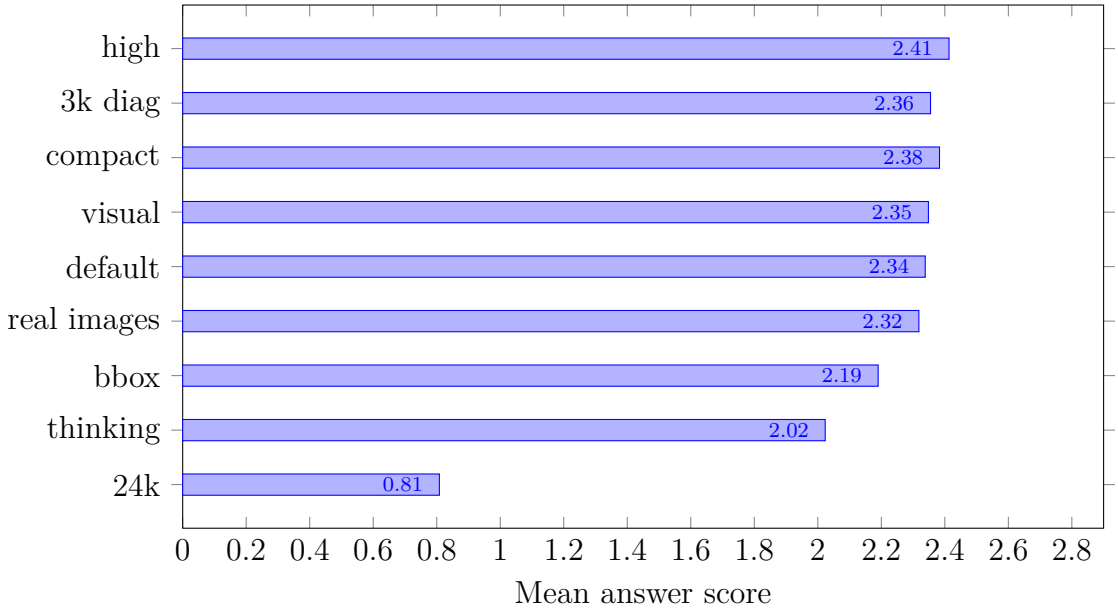


Figure 9.1: Answer scores of final 200-question preset runs

The horizontal bar chart compares high, compact, visual, default, real images, bbox, thinking, and 24k runs by mean score, showing that high has the highest mean while 24k fails operationally.

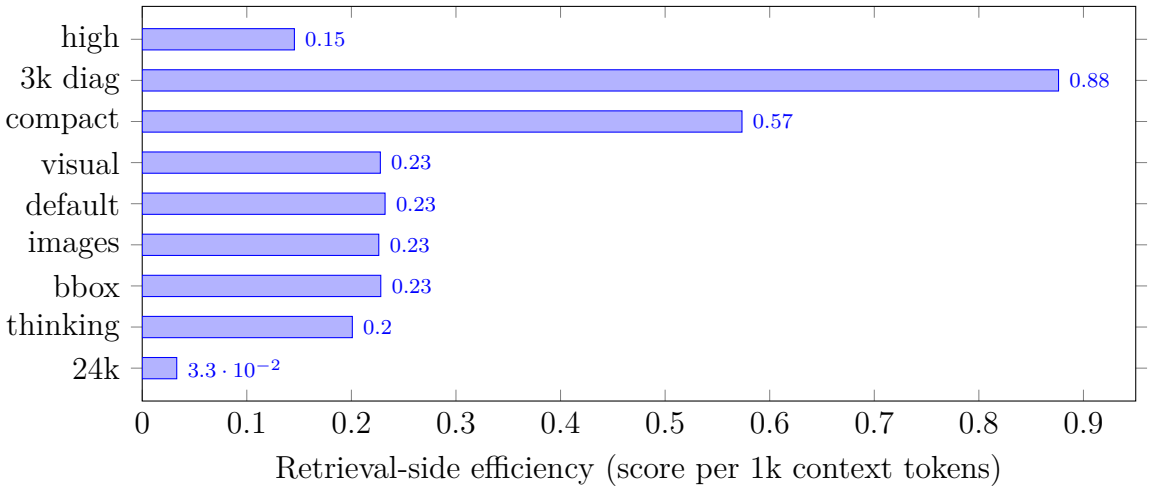


Figure 9.2: Token efficiency of final 200-question preset runs

The horizontal bar chart measures score per thousand retrieved-context tokens, showing that the 3k diagnostic setting is cheapest but lower quality, while compact is the deployable efficiency-oriented preset.

Figure 9.3: Effect of real image input on total token usage

9.7. Breakdown by Question Type

The 200-question breakdown explains why the final system is a set of presets rather than one universal mode. The `high` preset improves fact, procedure, cross-page, and multi-PDF questions, but it is not the best for every type. `medium` text remains strong on visual questions because many visual questions are answerable through captions, patched text, and table/image descriptions already inserted upstream. Naive visual retrieval helps multi-PDF comparison in this set, but it is slower and does not beat `high` overall. Repaired image input slightly helps some visual-type rows, but the mean score and total-token cost still make it unsuitable as the default.

Question type	Count	Default	High-recall	Compact	Visual-recall	Real images
Fact	41	1.797	1.919	1.927	1.813	1.821
Procedure	45	1.770	2.074	2.030	2.015	1.852
Table/specification	36	2.528	2.398	2.463	2.185	2.324
Visual	30	3.478	3.233	3.189	3.333	3.433
Cross-page	29	1.770	1.908	1.920	1.667	1.690
Multi-PDF comparison	15	3.178	3.467	3.267	3.933	3.511
Safety or notice	4	5.000	5.000	4.333	4.667	4.333

Table 9.9: Mean score by question type for selected final 200-question runs

The type breakdown supports exposing multiple presets rather than forcing all traffic through the highest-recall mode.

The high visual scores for text-only modes are not contradictory. Many visual questions ask about information that the pipeline has already transformed into text through patching, captioning, table extraction, or source metadata. For those questions, sending the original image again is redundant. The remaining visual failures are more likely to need localized image crops than full-image broadcasting. The thesis therefore recommends future localized image evidence rather than simply turning on image input for all questions.

9.8. Failure and Score Distribution

The final runs have many low and mid scores because the benchmark is intentionally hard: it includes multi-page procedures, cross-document comparisons, diagram references, and questions whose answer is absent unless the correct source and page are retrieved. The relevant comparison is not whether every preset reaches a high absolute score; it is how each preset changes the distribution of severe failures, useful answers, and operational errors.

Run	Score ≤ 1	(1,2]	(2,3]	(3,4)	≥ 4
`default text`	18	132	10	2	38
`high 16k`	17	131	14	4	34
`compact ratio0.4`	15	139	9	3	34
`default real images`	26	122	12	2	38
`visual naive w1`	18	131	8	3	40
`visual bbox w3`	26	129	9	5	31
`thinking on`	63	83	11	8	35
`very high 24k`	138	42	7	2	11

Table 9.10: Score-bucket distribution in selected final 200-question runs

Thinking and 24k context create many more severe failures.

Run	Usage missing	Image URLs	Dominant failure pattern in reviewed rows
`default text`	0	0	Retrieval failure is the largest class (113 rows), followed by partial or wrong answers and unsupported answers.
`default real images`	0	1378	Retrieval failure remains the largest class (108 rows); image input adds cost but does not remove the main retrieval bottleneck.
`visual naive w1`	0	0	Retrieval failure remains similar (110 rows); visual route helps some comparisons but does not transform the overall distribution.
`thinking on`	0	0	Retrieval failure remains high and answering-side errors grow; latency increases to 13.30 s.
`very high 24k`	124	0	Over-context serving failure dominates: 124 usage-missing cases and many empty or broken answers.

Table 9.11: Failure audit for key final runs

The rejected modes are excluded because they worsen failure profiles or operating constraints, not only because their mean score is lower.

The failure audit determines which experimental switches become runnable modes and which remain rejected controls. Thinking-on is rejected not only because its mean score is lower, but also because it creates more severe failures and much higher latency. Image input loses only 0.0200 mean score, but it adds 1378 image URLs and a large token increase while retrieval failure remains the largest class. The 24k context fails the current serving boundary with 124 usage-missing cases, so it is not exposed as a preset.

9.9. Qualitative Spot Checks

The final preset decision still relies on the 200-question aggregate scores, but a small manual spot-check helps explain what the score differences mean in concrete manual-QA cases. I selected rows where the aggregate result could be misunderstood: one high-recall win, one compact win, one visual-route change, one image-input failure, and one serving/answering failure. The spot-check did not relabel the benchmark. It compared selected archived rows against their canonical answers, evidence summaries, generated answers, pass-level notes, and run metadata.

High-recall helps when the question is about source selection. In `qa-0054`, the question asks which S3006 visual source should answer size questions and which should answer port-layout questions. The canonical answer separates the S3006 dimensions PDF from the S3006 port diagram. `default` scores 2.0000 and `compact` scores 1.6667 because they choose broader datasheet or user-manual sources instead of the two visual source

PDFs. `high` scores 4.0000: it retrieves both the dimensions and port evidence and gives the right mapping, although the answer still has a minor source-name imprecision. This example supports using `high` for hard manual review, but it also shows why the higher score is not treated as a perfect answer.

Compact can be better when the decisive evidence is small. In `qa-0013`, the question asks which S3006 ports are marked as uplink ports. The compact run answers "ports 5 and 6" from the port diagram and scores 5.0000 with 3565 retrieved-context tokens. The high run also answers correctly, but uses 16311 retrieved-context tokens and scores 4.6667. A manual reading of the row agrees with the score direction: when the needed fact is concentrated in one retrieved diagram caption, more context mainly adds cost and possible distraction rather than new evidence.

Visual recall changes the evidence mix, not only the amount of evidence. In `qa-0055`, the question asks which S3006 source is better for installation-sequence diagrams and which is better for prose installation and operation context. The text-only rows repeatedly choose the datasheet for diagrams and score around 2.3333. The visual-recall row retrieves the installation-method sheet and the switch user manual together, answers with the correct source split, and scores 5.0000. The same mechanism can hurt other questions: for `qa-0038`, a PoE pin-assignment table question, visual recall still gives the right text answer but the retrieved excerpt shown to the reviewer does not support it, so the row scores 2.0000. This is why visual recall remains an explicit route rather than the normal route.

Image input can add noise instead of solving a visual question. In `qa-0030`, the canonical answer is that the HDCVI Box Camera Installation Guide provides two pages of diagrammatic installation evidence. The real-image `default-with-image-input` row sends 11 image URLs to the answerer but answers "12 pages" and scores 1.0000. This is not a case where the model failed because images were absent; it is a case where image evidence was present but the answerer confused page count with the number of visual items. The row illustrates why the thesis rejects always-on image input and recommends future localized crops or smaller image sets instead.

Some rejected modes fail as systems, not only as scorers. In `qa-0042`, the canonical answer is the DH-HAC-HF3805G effective pixel count, 3840(H)×2160(V), 8MP. The default, compact, high, visual-recall, and image-input rows all answer correctly. The thinking-on row keeps the relevant context but produces no answer and scores 0.0000; the pass notes describe an answering failure despite available evidence. The 24k row fails even earlier: it records `usage_missing=1`, zero total tokens, and a serving error saying

that the requested token count exceeded the 32768-token model limit. This kind of row is the practical reason for treating thinking and 24k context as rejected controls, not just lower-scoring alternatives.

These examples do not replace the quantitative tables. They connect the tables to visible failure mechanisms: missing or diluted source selection, compact evidence sufficiency, route-dependent support, image-induced counting confusion, and serving-boundary failures. They also justify a cautious reading of individual automated scores. The reliable claim is the aggregate preset behavior plus recurring failure patterns, not that every single fractional score is a human-equivalent judgment.

9.10. Final Presets

The final interface contains six runnable modes. The thesis keeps the historical experimental names when discussing archived runs, while the engineering interface uses canonical operational names. Two image-input modes are diagnostic modes; `medium-with-image-input` is directly measured on the final 200-question set, while `low-with-image-input` is included as a runnable counterpart with derived cost estimates. The compatibility aliases are:

- `compact` maps to `low`.
- `default` maps to `medium`.
- `high-recall` maps to `high`.
- `compact-with-image-input` maps to `low-with-image-input`.
- `default-with-image-input` maps to `medium-with-image-input`.
- `visual-recall` maps to `medium-with-visual-recall`.

Preset	Engineering name	Use	Source	Mean	Avg context	Max context	Avg total	Max total	Latency
compact	low	Low-token batch or support mode	200QA	2.3833	4156.10	11279	4565.19	11585	2.79
default	medium	Conservative text baseline	200QA	2.3383	10071.58	11802	10588.75	13408	3.37
high-recall	high	Manual review and hard questions	200QA	2.4133	16584.94	17796	17198.90	19912	4.17
compact-with-image-input	low-with-image-input	Low text plus evidence images	Derived estimate	n/a	approx. 4156.10	approx. 11279	approx. 8200–8300	n/a	n/a
default-with-image-input	medium-with-image-input	Medium text plus evidence images	200QA	2.3183	10251.18	11944	14325.33	23442	4.65
visual-recall	medium-with-visual-recall	Visual/UI recall route	200QA	2.3483	10314.86	11922	10965.50	13612	8.43

Table 9.12: Final presets with observed or derived token budgets

Context tokens exclude image-token cost; total tokens include LLM usage when available.

The six presets are named by use case rather than by experiment id. Archive run ids are useful for audit, but they are not suitable as user-facing modes. A final interface should

provide stable names whose behavior is documented: `low`, `medium`, `high`, image-input variants, and `medium-with-visual-recall`. The older names remain useful as compatibility aliases and as labels for the experimental narrative, but the canonical deployment names are the low/medium/high family.

The exact quality parameters are summarized below.

Preset	Engineering name	Route	Visual bonus	k	Top-k	Cap	Score ratio	Images	Thinking
`compact`	`low`	text	none	150	10	10000	0.4	0	off
`default`	`medium`	text	none	80	20	10000	1.0	0	off
`high-recall`	`high`	text	none	150	80	16000	1.0	0	off
`compact-with-image-input`	`low-with-image-input`	text	none	150	10	10000	0.4	1	off
`default-with-image-input`	`medium-with-image-input`	text	none	80	20	10000	1.0	1	off
`visual-recall`	`medium-with-visual-recall`	text-visual-naive	page-naive	150	20	10000	1.0	0	off

Table 9.13: Core retrieval and answering quality parameters for the final presets

Shared defaults include RRF equal to 10 and answer max tokens equal to 8000.

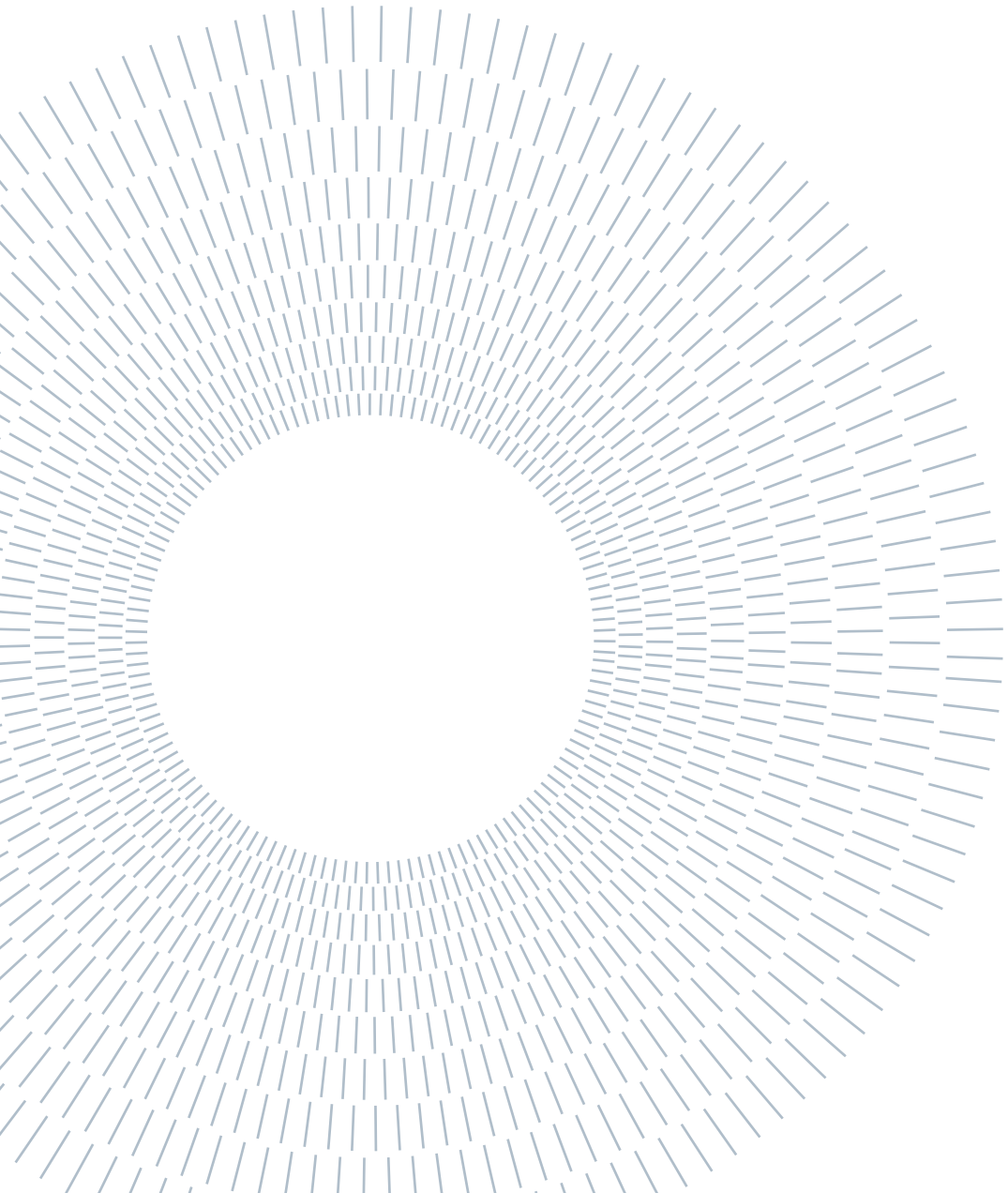
9.11. Preset Caveats

`default` corresponds to `medium` in the engineering interface and is a stable, inexpensive, and simple conservative baseline. `high-recall` corresponds to `high`; it has the best final score but is more suitable for manual review or difficult questions. `compact` corresponds to `low`. It is not a hard 4k cap; it keeps a 10k soft cap and achieves low average context through `top_k=10` and score-ratio pruning. It is a suitable choice for high-volume support, batch drafting, or other token-sensitive workflows. `compact-with-image-input` corresponds to `low-with-image-input` and is a derived engineering preset; it has not been separately validated on 200 questions, so its total-token range is estimated from observed image-input overhead. `visual-recall` corresponds to `medium-with-visual-recall`; it changes retrieval ranking but does not pass images to Qwen. `default-with-image-input` corresponds to `medium-with-image-input`; it passes evidence images to Qwen but does not change retrieval route. For reporting, the two switches are documented separately.

The final system keeps thinking off by default, avoids the 24k context cap under the current 32768-token serving limit, and does not make real image input the default answering path. If future work revisits image input, the better direction is to send fewer and more localized image crops rather than all recommended evidence images.

These caveats are part of the preset contract: they record the operating conditions for each mode. `high` can be overused for routine queries, `low` can be mistaken for a hard context cap, and `medium-with-visual-recall` can be mistaken for image input. The

chapter documents both preset behavior and preset limits.



10 | Discussion and Conclusion

10.1. What the Experiments Show

The experiments did not simply confirm the initial design. In v1, multimodal preprocessing looked like the most natural direction: small-icon patching and image captioning made missing visual evidence searchable, and visual retrieval exposed pages that plain text retrieval could miss. In the later v2 answer runs, however, the online part behaved less cleanly. Sending more images or more text to the answerer was often expensive, sometimes slower, and not reliably better.

The clearest positive result is still structural. Final answer quality improves when the pipeline uses section-aware chunks rather than the original large v1-style chunks or plain token windows. The selected v2 profile, 1500-token maximum, 150-token overlap, and 150-token minimum, is not a magic value; it is the setting that survived the 50-question search and then fit the 200-question preset validation without changing the artifact contract. The practical lesson is that sectioning has to be written into the pipeline before retrieval, not reconstructed after a bad answer.

Retrieval and answering then split into operating modes rather than one winner. The list below uses the engineering names used by the implementation; the historical experiment names remain compatibility aliases for audit:

- `medium` (historical `default`) for conservative interactive reliability.
- `high` (historical `high-recall`) for the best observed final quality.
- `low` (historical `compact`) for the best token efficiency.
- `medium-with-visual-recall` (historical `visual-recall`) for visual/UI-heavy search where latency is acceptable.
- `medium-with-image-input` (historical `default-with-image-input`) for diagnostic visual answering on the medium text route.
- `low-with-image-input` (historical `compact-with-image-input`) for diagnostic vi-

sual answering on the low-token route.

This distinction became important during the negative runs. Multimodal RAG in this project is not a binary choice between text and images. RAG-Flow uses visual models during ingestion to repair and describe evidence, but it requires stronger evidence before adding visual cost at query time. The ordinary online path stays text-first because the retrieved text already contains many image descriptions, table fragments, and source breadcrumbs.

Direct-PDF diagnostics add a useful but uncomfortable reference point. When Codex `gpt-5.5` with `xhigh` reasoning effort inspects the original PDFs directly, it scores substantially higher than the selected RAG preset answers under a content-only rescore. Restricting the same Codex setup to the archived high-RAG context improves over the original `high` Qwen answerer, but still remains far below direct PDF access. This means the source files often contain enough information; RAG-Flow loses part of it through parsing, visual preservation, chunking, retrieval, evidence selection, or answerer use of the evidence. The direct-PDF row is therefore a reference for recoverable source information, not a deployable online baseline.

The v3 text-backbone experiment adds a narrower control to this interpretation. It keeps the v2 chunks, answer prompt, answerer, and text-only online path fixed while changing only dense/sparse retrieval components. The best medium-budget tested backbone, `Qwen/Qwen3-Embedding-4B + Qdrant/bm25 + rrf_k=5`, improves the 200-question content-only score from 1.2633 to 1.3250 against the current medium-style text backbone. A stricter matched control against the original v2 `high` preset is less favorable: keeping `k=150`, `top_k=80`, 16k context, and `rrf_k=10`, the Qwen answer row decreases from 1.3567 with e5-large to 1.3200 with Qwen3-4B. The same new context helps Codex slightly, from 1.6842 to 1.7250, but the direct-PDF row in the same matched-control rescore batch remains much higher at 2.7617. Backbone choice matters, but dense-backbone replacement alone does not solve the high-recall system weakness.

10.2. Answers to the Research Questions

The first research question asked how technical-manual PDFs can be converted into structured, traceable multimodal evidence. The answer is the staged artifact contract: MinerU parsing, outline-based sectioning, small-icon patching, image captioning, section-aware chunking, Qdrant indexing, retrieval, and answering. Each block and chunk carries source identity, page information, and breadcrumbs so that final answers can be audited.

The second question asked which first-version parameters remain valid. The final system keeps v1 visual-enhancement baselines for patching and captioning: `dpi=250` for icon repair and `max_image_side=2048` with 2000 context tokens for image captioning. It also keeps the v1 conclusion that thinking and page-image answering should not be defaults. v2 changes the chunking and retrieval defaults because the multi-document answer-centered evaluation gives stronger evidence than the earlier single-manual coverage study.

The third question asked whether section-aware chunking improves final answer quality. The answer is yes under the completed v2 runs: `auto 1500/150/150` scores 2.5400 on the 50-question chunking search, compared with 2.2867 for the large v1-style `5000/500/200` profile and 2.2933 for pure token-window `1500/150/150`. Sectioning is not only metadata for display; it improves chunk boundaries.

The fourth question asked which retrieval parameters should be used. The conservative text baseline is the historical `default` preset, with engineering name `medium`; it uses `k=80`, `top_k=20`, `rrf_k=10`, a 10k context cap, and no score-ratio pruning. `high-recall` has the engineering name `high` and uses `k=150`, `top_k=80`, and 16k. `compact` has the engineering name `low` and uses `k=150`, `top_k=10`, a 10k cap, and `score_ratio=0.4`. These settings are selected by answer quality, token cost, and latency rather than by retrieval hit-rate alone. The v3 robustness check further shows that `Qwen/Qwen3-Embedding-4B + BM25 + rrf_k=5` is a stronger medium-style text-backbone candidate when the larger embedding model and 2560-dimensional index are acceptable, but the matched high-control does not support applying the same replacement to `high`.

The fifth question asked whether visual retrieval, image input, or thinking should be default. The answer is no for the default path. Naive visual retrieval remains available as `visual-recall`, corresponding to `medium-with-visual-recall`, but it is slower. Real image input reaches the model but does not improve mean score. Thinking decreases score and increases latency. These are supported modes or rejected modes, not defaults.

The sixth question asked which final presets should be provided. The final interface provides six engineering names: `low`, `medium`, `high`, `low-with-image-input`, `medium-with-image-input`, and `medium-with-visual-recall`. The image-input modes are diagnostic rather than default modes, and the low-image variant is reported with derived cost estimates rather than a separate 200-question validation row. Historical experiment names remain useful for audit and are retained as compatibility aliases, but the user-facing contract is the engineering-name family.

Research question	Final answer in one sentence
Structured multimodal evidence	Use a staged artifact contract from parsing through answering, with source identity written before enrichment.
Inherited v1 baselines	Keep v1 patching and captioning quality settings, but retest chunking/retrieval/answering under v2 scoring.
Section-aware chunking	Yes: `auto 1500/150/150` outperforms large v1-style and token-window controls.
Retrieval defaults	Use text-only `k=80/top_k=20/10k` for `default`/`medium`, with separate `high-recall`/`high` and `compact`/`low` modes; v3 identifies `Qwen3-4B + BM25 + rrf5` as a stronger medium-style candidate, not a proven `high` replacement.
Visual, image, and thinking defaults	Do not enable them by default; keep visual/image modes explicit and diagnostic.
Final presets	Provide six documented engineering modes and retain historical experiment names as compatibility aliases.

Table 10.1: Condensed answers to the research questions

10.3. Why Some Candidate Modes Are Rejected

Several configurations looked attractive before the 200-question runs. The rejection reasons are not all the same.

24k context is rejected because it crosses the stable serving boundary. The run records 124 usage-missing cases. Some rows did not merely receive weaker answers; they returned no usable answer after the server rejected the prompt as longer than the 32768-token model limit. For example, in the DH-HAC-HF3805G effective-pixel question, the ordinary text rows answer 3840(H)×2160(V), 8MP, while the 24k row records zero total tokens and an over-limit error.

Thinking-on is rejected because it is unstable in this evidence-bound setting. It sometimes helps, but it also creates empty answers on rows that the text-only modes answer correctly. In the same effective-pixel question and in the S3006 uplink-port question, the retrieved context contains the answer, but the thinking-on row produces no answer and receives 0.0000. The problem is therefore not only average score; it is a new answering failure mode.

Default image input is rejected because real images reach Qwen but do not improve mean score. The clearest checked failure is the HDCVI Box Camera Installation Guide

page-count question: the image-input row sends 11 image URLs, but answers 12 pages instead of the canonical two pages. The current image selection is too broad for this type of visual counting question. Future image input should localize crops or pass a much smaller top-k set.

BBox visual attribution is rejected because page-level ColPali signals are not enough to reliably assign fine-grained chunk visual relevance. Naive page broadcast is more robust in the current implementation, although it is slower and still not a default.

These rejected modes define preset boundaries. A setting is not excluded only because it loses a leaderboard row; it is excluded when the run shows a failure profile that would be hard to explain to a user of the system.

10.4. Deployment Strategy

The final presets are a deployment strategy, not a leaderboard where the top row replaces every other row. The correct mode depends on the operational situation: latency budget, token budget, whether the user is doing interactive troubleshooting, and whether visual evidence is likely to matter.

Situation	Recommended preset	Rationale
High-volume support or ticket drafting	`low`	Comparable final quality with much lower average retrieved context; best token efficiency for repeated daily use.
Conservative interactive manual QA	`medium`	Stable text-only baseline with simple deployment. It avoids visual-route and image-input overhead.
Difficult question or manual review	`high`	Highest final 200-question mean score; useful when extra context is acceptable.
Batch preview, low-cost screening, or token-constrained use	`low`	Strong final score with much lower average retrieved context and best token efficiency.
Question likely about UI layout, diagram location, or visually distinctive page	`medium-with-visual-recall`	Uses ColPali page recall to change retrieval ranking, but keeps answering text-only.
Question explicitly requires looking at evidence images	`medium-with-image-input` `low-with-image-input`	Sends retrieval-provided evidence images to Qwen; diagnostic rather than default because measured quality does not improve on average.
Large-context stress testing	none; do not use 24k as a preset	The 24k run fails under the current serving limit and should not be provided as a user-facing mode.

Table 10.2: Recommended deployment strategy for the final presets

Under this strategy, `low` is not simply a weaker default. It is the preferred mode for high-volume support workflows, quick exploration, many repeated questions, or lower token

spend. Likewise, `high` is not automatically the default even though it has the best mean score, because its context cost is higher and its improvement is not large enough to justify using it for every interactive query.

The deployment strategy can be implemented as a simple escalation rule. Start with `low` for high-volume support or ticket-drafting workflows, and use `medium` when a conservative interactive baseline is preferred. If the answer is incomplete, unsupported, or the user explicitly asks for maximum evidence, rerun with `high`. If the question mentions a drawing, port layout, screenshot, icon, or visual location, try `medium-with-visual-recall`. Only pass images to the answerer when the application or reviewer explicitly wants the model to inspect retrieved images.

10.5. Agentic Baseline and Enterprise-Scale RAG Value

The direct-PDF diagnostic baseline clarifies the difference between indexed retrieval and agentic document navigation. The direct-PDF row is a reference condition for recoverable source information rather than a conventional online RAG baseline. Codex `gpt-5.5` with `xhigh` reasoning effort can decide which files to inspect, choose search terms, revisit pages, spend more reasoning on ambiguous questions, and recover evidence from raw PDF pages that the fixed RAG pipeline does not provide to the answerer. That search strategy is part of the intervention. The result shows what information is still present in the source documents, but production deployment still needs indexed retrieval for controlled latency, token budget, and auditability.

The timing evidence makes the distinction concrete. On the 200-question diagnostic, the deployed `high` RAG answer path takes 4.17 seconds per question from question input to final answer. Codex over archived high-RAG context takes 15.02 seconds per question, while Codex direct PDF takes 46.81 seconds per question. The matched Qwen3-4B high context is slower than the original Qwen RAG row when answered by Qwen, 9.76 seconds per question, but Codex over that archived context takes 13.58 seconds per question. Direct PDF is still about 11.2× slower than the deployed `high` RAG answer path, and the serial 200-question runtime increases from 13.89 minutes to 156.04 minutes. In addition, the direct-PDF experiment does not provide a reliable bounded online token count, because the Codex CLI does not provide complete token accounting for every internal file inspection, search, and context construction step. This limitation matters for deployment because predictable latency and predictable token budgets are part of an enterprise

support system requirement.

Under this interpretation, the value of RAG is less about beating an agent with stronger reasoning ability on a small benchmark and more about moving expensive document processing into offline ingestion. RAG turns large manuals into indexed and auditable evidence and bounds the amount of online context sent to the answering model. This matters more as the corpus grows. With a few manuals, an agent can often afford to search the original PDFs at answer time. With hundreds, thousands, or tens of thousands of PDFs, asking an agent to rediscover the relevant pages for every ticket can produce unstable latency, large and hard-to-predict context usage, and search traces that are difficult to audit. RAG-Flow's presets instead provide a controlled online contract: `low`, `medium`, and `high` define explicit context budgets; visual recall and image input remain explicit escalations; and every returned evidence item carries source and page metadata.

The direct-PDF row exposes the clearest limitation of the current implementation. It shows that the present RAG pipeline loses important evidence in some cases. The repair targets are concrete: page-level visual preservation, better handling of fragmented diagrams, stronger retrieval over captions and section text, confidence-aware escalation, and answerer-side use of retrieved evidence. The context-only Codex diagnostic is useful here because it shows that a stronger answerer can extract more value from the same archived RAG context than the deployed answerer, while still remaining below direct PDF access. The remaining gap is shared between retrieval/evidence exposure and answering capability.

A practical future architecture is agent-guided RAG rather than either fixed RAG alone or unrestricted direct-PDF search. In this design, RAG-Flow becomes a set of bounded tools available to an agent. The agent can start with `low` for cheap high-volume support, escalate to `medium` or `high` when confidence is low, call `medium-with-visual-recall` when the question appears diagram- or layout-dependent, and request image-input or direct page/PDF inspection only as an exceptional fallback. The agent supplies adaptability, while RAG supplies scale, cost control, source grounding, and repeatability.

The operational boundary is therefore separate from the preset ranking. Fixed RAG is faster and more predictable. Direct-PDF agents provide a source-information reference point, but they are slower and less bounded. A next system should combine the two by letting an agent select among RAG presets under explicit latency, token, and audit constraints.

10.6. Threats to Validity

The first threat is dataset scope. The 200-question final set covers 14 technical PDFs and diverse question types, but it remains domain-specific. New vendors, languages, scanned-only PDFs, or manuals with weak outlines may shift the optimal chunking or retrieval parameters.

The second threat is scorer dependence. The final protocol is stronger than automatic coverage scoring because it uses three separate Codex review passes against canonical answers, gold-evidence summaries, and retrieved-context excerpts. Still, it is not the same as a blinded human panel. The thesis mitigates this by saving pass-level notes, using paired comparisons, reporting failure types rather than only averages, and checking representative rows qualitatively. The spot-checks improve interpretation, but they do not remove the need for a future blinded human scoring audit.

The third threat is serving configuration. The negative 24k result is partly a model-serving boundary: Qwen plus an 8000-token completion budget under a 32768-token context limit cannot reliably accept that much retrieved text. A future model with a larger stable context could change the high-recall cap.

The fourth threat is image handling. The current image-input experiments pass evidence images selected by retrieval metadata. They do not crop local regions or use calibrated visual patch attribution. The negative image-input result is therefore a statement about the current image pipeline, not a general conclusion about every possible image-evidence design.

The fifth threat is language and vendor scope. The current corpus is English-language vendor documentation in the security-device and enterprise-software domain. Other languages, scanned manuals, CAD-heavy manuals, or documents with weaker outlines may shift the balance among dense retrieval, sparse retrieval, visual retrieval, and section-aware chunking.

The sixth threat is metric interpretation. The final mean scores are useful for comparing configurations within this benchmark, but they are not a universal quality scale. The benchmark is intentionally hard and includes many questions that require multi-evidence synthesis. The absolute value of a mean score should be interpreted alongside high-quality rate, low-score rate, failure types, and qualitative examples.

The seventh threat is model asymmetry in the direct-PDF diagnostic baseline. The direct-PDF row uses Codex `gpt-5.5` with `xhigh` reasoning effort and agentic PDF navigation,

whereas the deployed RAG answerer uses the RAG-Flow retrieval payload and the configured Qwen answering service. The context-only Codex diagnostic partly controls this asymmetry by using the same Codex setup while freezing the evidence to archived RAG context. Its score sits between the original RAG answerer and direct PDF access. The diagnostic therefore has a narrower role: it shows that a strong model can use the archived context better, while still recovering substantially more from the original PDFs than the current RAG pipeline provides to the answerer.

The eighth threat is v3 selection scope. The text-backbone experiment fixes the v2 chunks, captions, answer prompt, and answerer. This makes the dense/sparse comparison controlled, but it also means the experiment cannot measure improvements from different chunking, localized image preservation, reranking, or agentic retrieval. The v3 winner is a better tested medium-budget text backbone under fixed conditions, not a complete new RAG system. The matched v2-high control further shows that the same dense replacement does not automatically transfer to the high-recall preset.

The ninth threat is external-framework comparison. The experiments study the internal RAG-Flow pipeline and preset strategy rather than a horizontal ranking against complete RAG frameworks such as LlamaIndex, Haystack, Dify, or RAGFlow. A fair comparison would need to control the PDF parser, chunking input, embedding model, sparse retrieval, reranking, answer prompt, answer model, and archived retrieved context. Without those controls, a framework comparison would mix architecture, parser quality, model choice, and tuning effort. External full-framework benchmarking is treated as future work rather than as a main validity claim.

10.7. Future Work

The most useful next steps are:

- Localized image evidence: crop bbox-level regions instead of passing full images.
- Calibrated ColPali attribution: map visual patch hits back to PDF geometry before assigning chunk-level visual scores.
- Human scoring audit: sample Codex-scored answers for blind human review.
- Cross-domain validation: repeat final presets on manuals outside the Dahua/security-device source set.
- External-framework benchmarking: compare against LlamaIndex, Haystack, Dify, and RAGFlow only under controlled parser, chunking, retrieval, and answerer condi-

tions.

- Agent-guided RAG presets: organize `low`, `medium`, `high`, visual recall, image input, and direct page/PDF fallback as bounded tools selected under latency, token, and audit constraints.

There are also engineering extensions that follow directly from the thesis. First, retrieval could add confidence-aware escalation: if `medium` returns low evidence scores, high source ambiguity, or a short context with weak support, the system could automatically rerun `high`. Second, image-input presets could be made adaptive rather than manual: only images with high image-answering policy confidence and high localized visual relevance would be sent. Third, the review archive could be used to train or calibrate a lightweight reranker or failure predictor, as long as the resulting model remains auditable.

10.8. Final Quality Checklist

Before treating RAG-Flow as a final system, the following checklist summarizes the quality conditions established by the experiments:

- The final source corpus and gold answers are anchored to original PDFs and rendered pages.
- Sectioning writes source identity before LLM enrichment.
- Patching and captioning operate on current sectioned artifacts.
- Chunking uses section-aware `1500/150/150` and triggers text reindexing when chunk text changes.
- Retrieval presets are selected by answer score, token cost, latency, and failure profile.
- Answering uses thinking off and no image input by default.
- Image-input rows are interpreted only after verifying real image URLs.
- 24k context is not provided under the current serving limit.

10.9. Conclusion

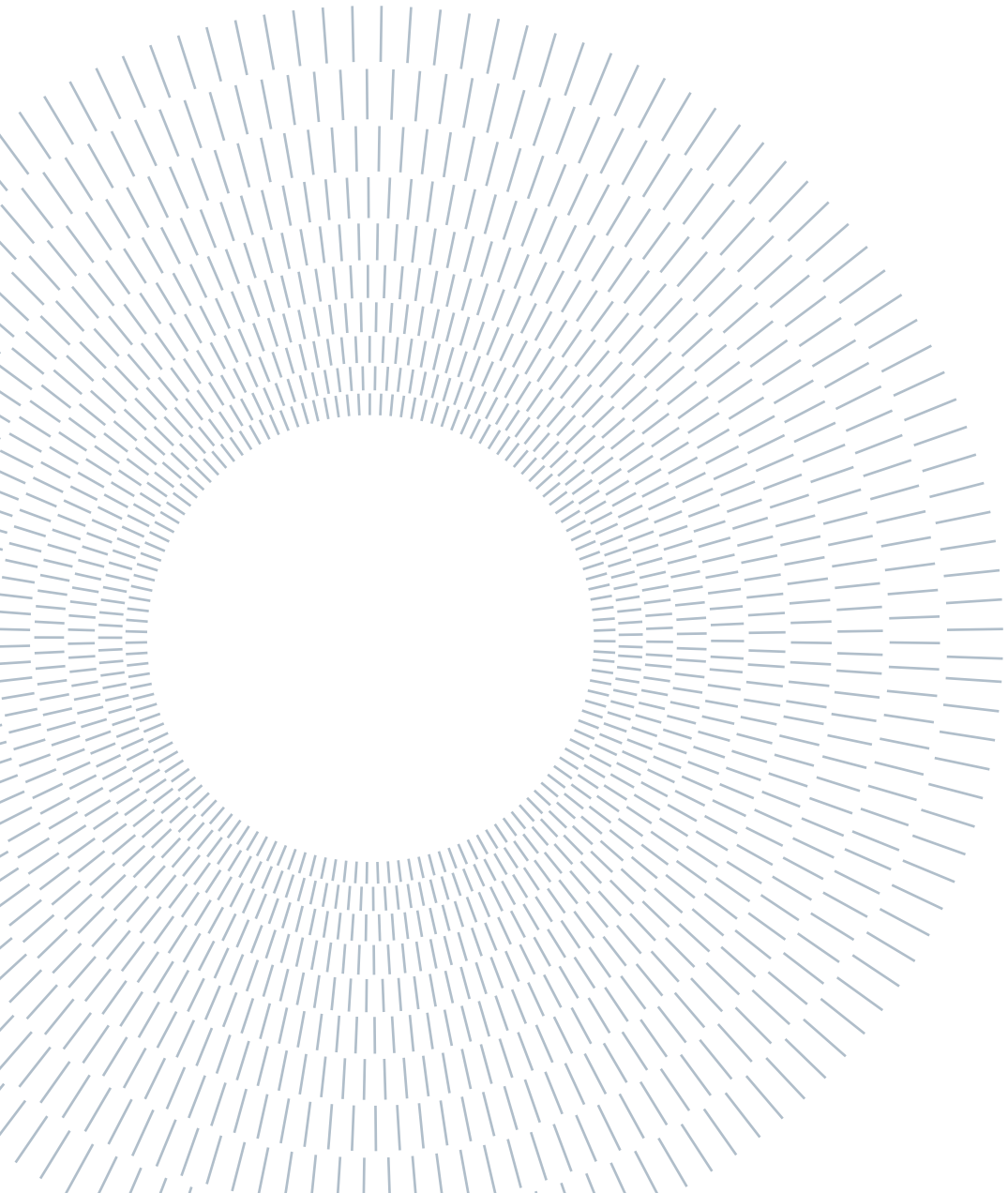
RAG-Flow is a working multimodal RAG pipeline for technical manuals, but the experiments are most useful where they make the system less ambitious. The v1 runs show that visual preprocessing is worth keeping: patching and captioning turn icons, drawings, and screenshots into text-like evidence that can be retrieved and audited. The v2 runs

then show that this does not justify sending every possible visual artifact to the answerer. The stronger result is the artifact contract itself: source identity, section paths, page references, chunks, retrieval outputs, answers, and review notes remain connected through the pipeline.

The final preset set is therefore pragmatic. `medium` is the conservative text-only path. `high` is the escalation path when extra context is acceptable. `low` is the cheaper path for repeated or high-volume use. Visual recall and image input remain available, but they are explicit modes because the negative rows are real: 24k can hit the serving limit, thinking-on can return empty answers, and image input can confuse image-item counts with page counts.

V3 narrows one possible explanation without solving the whole problem. The `Qwen3-4B` plus BM25 and `rrf5` row is a stronger tested medium-style backbone, but the matched high-control does not show the same improvement for the deployed high-recall row. The direct-PDF diagnostic makes the remaining gap clearer: more information is recoverable from the original PDFs than the current fixed RAG path exposes to Qwen. That gap is shared by parsing, visual preservation, retrieval, evidence selection, and the answerer.

For deployment, the safest rule is still simple: start cheap when the task allows it, escalate when evidence is weak, and keep image-heavy modes explicit. This is not a claim that the current presets are universally optimal. It is a documented boundary around the system that was actually run, measured, and checked against its failures.



A | Appendix: Exact Presets and Additional Tables

A.1. Reference Material

The thesis uses the v1 and v2 thesis generations, the v3 robustness-check archive, and the experiment artifacts listed below. The table records the role of each source: v1 is the system baseline, v2 is the final preset-validation layer, and v3 is the retrieval-backbone robustness check.

Reference root	Material used	Role in the thesis
`https://github.com/ziyan-c/RAG-Flow`	Public RAG-Flow code repository	Provides the code-availability reference for the pipeline implementation; large experiment artifacts and source PDFs are documented separately.
`thesis-v1/00-overview`	End-to-end v1 timing and overview text	Establishes the first complete system baseline and runtime profile.
`thesis-v1/01-environment`	Remote machine, Python, SGLang, and package-environment notes	Documents the serving and reproducibility context behind the first complete pipeline.
`thesis-v1/02-parsing-mineru`	MinerU parsing command, output structure, and parsing-quality notes	Supplies the detailed parsing contract summarized in the final system chapter.
`thesis-v1/03-sectioning`	PDF-outline sectioning discussion	Preserves the original section-recovery motivation.
`thesis-v1/04-patching`	Patching quality and throughput CSVs; stage text	Supplies the inherited `dpi=250` quality baseline and runtime bottleneck analysis.
`thesis-v1/05-captioning`	Captioning context, image-size, concurrency, and speedup data	Supplies the inherited `max_image_side=2048` and 2000-token caption context baseline.
`thesis-v1/06-chunking-indexing`	Original large-chunk and indexing rationale	Defines the v1 high-recall baseline retested in v2.
`thesis-v1/07-retrieval-serving`	v1 retrieval coverage and serving constraints	Provides the original retrieval baseline and ColPali serving caveats.
`thesis-v1/08-end-to-end-answering`	Main 50-question answering results and appendix checks	Supports the no-default-thinking and no-default-page-image conclusions.
`thesis-v2/00-overview` to `thesis-v2/09-answering`	v2 chapter notes and experiment records	Supplies the final stage artifact contracts, scoring protocol, chunking, retrieval, answering, and preset decisions.
`thesis-v2/experiments/v2-final`	Consolidated run JSONs, answering runs, CSV tables, and figures	Supplies all v2 quantitative results used in the final tables.
`thesis-v3/experiment-plan.md`	Retrieval-backbone experimental design	Defines the v3 dense/sparse, vector-size, fusion, RRF, and content-only scoring protocol.
`thesis-v3/experiments` and `thesis-v3/*.md`	V3 run manifests, Qwen answers, Codex scoring, matched high-control, and summaries	Supplies the retrieval-backbone robustness results and the matched v2-high interpretation check.
`qa-goldset`	Source PDF inventory, 200-question gold set, Codex gold-set review	Supplies source-corpus counts, evidence audit, question-type distribution, and review status.
`source-pdfs`	The 14 original technical PDFs	Defines the ground-truth document corpus for final evaluation.

Table A.1: Reference material used for the thesis

Together, these materials provide the v1 system evidence, v2 final validation, v3 retrieval-backbone checks, and gold-set audit.

A.2. Detailed Experiment Archive

The main methodology chapter keeps only the research-role overview for the three experiment groups. The following table preserves the complete parameter view, including corpus scope, scoring inputs, fixed baselines, and varied or validated parameters.

Phase	Corpus and question set	Scoring and measurements	Fixed baseline inputs	Main parameters varied or validated	Role in the thesis
v1 baseline	One main technical-manual pipeline; 220 reviewed retrieval questions; 50 answering questions; hard visual/thinking appendices.	Stage-specific review: patching quality, patching throughput, captioning throughput and grounding, retrieval MRR/coverage, automatic answering score, tokens, and latency.	MinerU parsing; PDF-outline section recovery; small-icon repair; text index plus optional page-level ColPali index.	Patching `dpi=250`, `batch_size=512`, checkpoint interval 1; caption `max_image_side=2048`, `max_context_tokens=2000`; early text/dense/sparse/visual routes; rendered-page answering and thinking checks.	Establishes end-to-end feasibility and the quality baselines inherited by v2.
v2 final validation	14 source PDFs; 50-question search set; 200-question final validation set; 88 answer-bearing runs; 6001 generated answers.	Three separate Codex 5.5 extra-high 0–5 reviews per answer; the main v2 scoring input contains the question, canonical answer, candidate answer, gold-evidence summary, and a retrieved-context excerpt clipped by the scoring script with a default 6000-character limit; reported metrics include mean score, high/low rates, context tokens, total tokens, latency, and failure notes.	Sectioned/patched/captioned JSON; Qdrant text schema; dense `intfloat/multilingual-e5-large` 1024-d cosine vector; sparse `Qdrant/bm25` with IDF modifier; page visual model `vidore/colpali-v1.3-merged`.	Chunking `auto 1500/150/150`; context cap; retrieval `k`; final `top_k`; RRF constant; score-ratio pruning; route mode; visual bonus/weight; image input; output budget; thinking; final preset aliases.	Provides the main answer-centered evidence for final chunking, retrieval, answering options, rejected modes, cost/latency trade-offs, and deployable presets.
v3 retrieval-backbone check	Fixed v2 corpus, chunks, prompt, and answerer; 50-question screening; 200-question confirmation; matched v2-high control on the same 200 questions.	Content-only Codex 5.5 extra-high review using only question, canonical answer, and candidate answer; Qwen tokens, retrieval latency, indexing time, collection size, and answer latency where available.	Final v2 chunk profile `1500/150/150`; text-only route; no image input; thinking off; same Qwen answerer and answer prompt; matched-control high budget fixes `k=150`, `top_k=80`, 16k context cap, and `rrf_k=10`.	Dense candidates including current e5 and Qwen3-Embedding variants; dense vector size including 1024, 2560, and 4096 rows; sparse candidates including BM25 and alternatives; dense-only, sparse-only, fused route, RRF sweep, and matched high-recall dense replacement.	Tests whether answer weakness is explained by the text-retrieval backbone alone; the best v3 row helps modestly but does not close the direct-PDF gap.

Table A.2: Detailed V1, V2, and V3 experimental-parameter archive

A.3. V2 Experiment Archive Inventory

The v2 final archive is stored as explicit tables, run manifests, per-question retrieval outputs, Qwen responses, and review files. The thesis uses the consolidated tables for main quantitative reporting and keeps the run directories as the audit trail.

CSV table	Data rows	Contents
`run_summary.csv`	88	Consolidated answer-bearing run summary with parameters, scores, tokens, latency, route, thinking, image flag, and indexing timings.
`chunking_key_results.csv`	8	Key coarse chunking rows used for the chunking figure and summary.
`retrieval_key_results.csv`	8	Representative retrieval rows used for default, compact, tiny, visual, and high-context comparisons.
`answering_max_token_results.csv`	5	Output-token sweep from 1000 to 8000 tokens.
`answering_feature_results.csv`	4	Text, real-image, thinking, and max8000 feature comparisons.
`answering_fixed_image_summary.csv`	2	Repaired 50-question image-input audit against text fixed-index baseline.
`final200_results.csv`	11	Final 200-question validation runs.
`final200_images_fixed_summary.csv`	1	Repaired 200-question image-input summary.

Table A.3: V2 final CSV inventory

These tables are backed by per-run directories under the v2 final answering-runs archive.

Group	Runs	Answers	Avg mean	Best run	Best	Worst run
Answering feature	6	300	2.3978	`answering-feature-images-on`	2.5800	`answering-feature-images-thinking-on`
Answering max tokens	5	250	2.5173	`answering-max-max8000`	2.5667	`answering-max-max4000`
Chunking	15	750	2.3898	`chunk-coarse-m1500-o150-n150`	2.5400	`chunk-coarse-m3000-o300-n200`
Retrieval sweep	37	1850	2.3942	`retrieval-sweep-cap10k`	2.5467	`retrieval-sweep-ratio0p0`
Retrieval interaction	13	650	2.4492	`retrieval-interaction-cap10-rk150-top20-naive-w1`	2.5933	`retrieval-interaction-cap10-rk150-top20-bbox-w3`
Final 200QA	11	2200	2.1689	`final200-high-context-16k`	2.4133	`final200-high-context-24k`
Smoke	1	1	1.0000	`smoke-fast-answer-only`	1.0000	`smoke-fast-answer-only`

Table A.4: V2 run-group summary from the consolidated archive

The low final200 average is pulled down by deliberately included rejected stress cases such

as 24k context and thinking-on.

A.4. Final 200-Question Run Archive

Run id	Mean	High	Low	Avg ctx	Avg total	Latency	Decision
`final200-high-context-16k`	2.4133	0.170	0.740	16584.94	17198.90	4.17	Highest final score; retained as `high-recall` (engineering `high`).
`final200-default-images`	2.3833	0.175	0.755	10071.58	10588.75	3.37	Historical zero-image audit; not a valid image-input run.
`final200-compact-ratio0p4`	2.3833	0.170	0.770	4156.10	4565.19	2.79	Best token-efficiency preset; retained as `compact` (engineering `low`).
`final200-tiny-cap3-top5`	2.3550	0.150	0.775	2687.77	3024.95	2.58	Diagnostic low-context lower bound; not provided as a preset.
`final200-visual-naive-w1`	2.3483	0.200	0.745	10314.86	10965.50	8.43	Retained as `visual-recall` (engineering `medium-with-visual-recall`).
`final200-default-text`	2.3383	0.190	0.750	10071.58	10588.75	3.37	Stable conservative baseline.
`final200-default-images-fixed`	2.3183	0.190	0.740	10251.18	14325.33	4.65	Valid image-input diagnostic; not default.
`final200-compact-top10`	2.2967	0.165	0.765	5823.84	6274.21	2.92	Dominated by score-ratio compact; archived only.
`final200-visual-bbox-w3`	2.1900	0.155	0.775	9599.53	10413.93	8.27	Rejected; bbox visual attribution is not stable enough.
`final200-thinking-on`	2.0233	0.175	0.730	10071.58	11930.10	13.30	Rejected; lower quality and much higher latency.
`final200-high-context-24k`	0.8083	0.055	0.900	24531.97	9293.99	2.73	Rejected; 124 usage-missing cases under current serving limit.

Table A.5: Complete final 200-question archive table

The main text interprets the same rows by preset family, token efficiency, and failure mode.

A.5. Exact Environment Blocks

A.5.1. Default

```
RAG_FLOW_RETRIEVAL_ROUTE_MODE=text
RAG_FLOW_RETRIEVAL_VISUAL_BONUS=none
RAG_FLOW_RETRIEVAL_K=80
RAG_FLOW_FINAL_TOP_K=20
RAG_FLOW_RRF_K=10
RAG_FLOW_RETRIEVAL_MAX_CONTEXT_TOKENS=10000
RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO=1.0
RAG_FLOW_ANSWER_MAX_TOKENS=8000
RAG_FLOW_ANSWER_ENABLE_THINKING=0
```

```
RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=0
```

A.5.2. High Recall

```
RAG_FLOW_RETRIEVAL_ROUTE_MODE=text
RAG_FLOW_RETRIEVAL_VISUAL_BONUS=none
RAG_FLOW_RETRIEVAL_K=150
RAG_FLOW_FINAL_TOP_K=80
RAG_FLOW_RRF_K=10
RAG_FLOW_RETRIEVAL_MAX_CONTEXT_TOKENS=16000
RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO=1.0
RAG_FLOW_ANSWER_MAX_TOKENS=8000
RAG_FLOW_ANSWER_ENABLE_THINKING=0
RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=0
```

A.5.3. Compact

```
RAG_FLOW_RETRIEVAL_ROUTE_MODE=text
RAG_FLOW_RETRIEVAL_VISUAL_BONUS=none
RAG_FLOW_RETRIEVAL_K=150
RAG_FLOW_FINAL_TOP_K=10
RAG_FLOW_RRF_K=10
RAG_FLOW_RETRIEVAL_MAX_CONTEXT_TOKENS=10000
RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO=0.4
RAG_FLOW_ANSWER_MAX_TOKENS=8000
RAG_FLOW_ANSWER_ENABLE_THINKING=0
RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=0
```

A.5.4. Compact With Image Input

```
RAG_FLOW_RETRIEVAL_ROUTE_MODE=text
RAG_FLOW_RETRIEVAL_VISUAL_BONUS=none
RAG_FLOW_RETRIEVAL_K=150
RAG_FLOW_FINAL_TOP_K=10
RAG_FLOW_RRF_K=10
RAG_FLOW_RETRIEVAL_MAX_CONTEXT_TOKENS=10000
RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO=0.4
RAG_FLOW_ANSWER_MAX_TOKENS=8000
RAG_FLOW_ANSWER_ENABLE_THINKING=0
```

```
RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=1
```

A.5.5. Visual Recall

```
RAG_FLOW_RETRIEVAL_ROUTE_MODE=text-visual-naive
RAG_FLOW_RETRIEVAL_VISUAL_BONUS=page-naive
RAG_FLOW_VISUAL_WEIGHT=1.0
RAG_FLOW_RETRIEVAL_K=150
RAG_FLOW_FINAL_TOP_K=20
RAG_FLOW_RRF_K=10
RAG_FLOW_RETRIEVAL_MAX_CONTEXT_TOKENS=10000
RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO=1.0
RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=0
RAG_FLOW_INDEX_MODE=both
RAG_FLOW_INDEX_VISUAL_DPI=200
RAG_FLOW_INDEX_VISUAL_BATCH_SIZE=8
RAG_FLOW_ANSWER_MAX_TOKENS=8000
RAG_FLOW_ANSWER_ENABLE_THINKING=0
```

A.5.6. Default With Image Input

```
RAG_FLOW_RETRIEVAL_ROUTE_MODE=text
RAG_FLOW_RETRIEVAL_VISUAL_BONUS=none
RAG_FLOW_RETRIEVAL_K=80
RAG_FLOW_FINAL_TOP_K=20
RAG_FLOW_RRF_K=10
RAG_FLOW_RETRIEVAL_MAX_CONTEXT_TOKENS=10000
RAG_FLOW_RETRIEVAL_MIN_SCORE_RATIO=1.0
RAG_FLOW_ANSWER_MAX_TOKENS=8000
RAG_FLOW_ANSWER_ENABLE_THINKING=0
RAG_FLOW_RETRIEVAL_FINAL_OUTPUT_IMAGES=1
```

A.6. First-Version Answering Appendix

Run	Context	Images	DPI	Score	Usable	Tokens
`text_ctx10k`	10000	0	200	4.18	0.78	10447.50
`text_ctx16k`	16000	0	200	4.20	0.82	16415.82
`img_ctx10000_dpi200_top1`	10000	1	200	3.96	0.70	14167.22
`img_ctx10000_dpi200_top2`	10000	2	200	4.10	0.74	18034.24
`img_ctx10000_dpi200_top3`	10000	3	200	4.14	0.82	21880.26
`img_ctx10000_dpi250_top2`	10000	2	250	4.08	0.76	22278.54
`img_ctx16000_dpi200_top3`	16000	3	200	4.10	0.80	27786.42
`img_ctx16000_dpi250_top3`	16000	3	250	4.02	0.70	34131.46

Table A.6: Representative first-version answering rows retained for comparison

A.7. Historical Diagnostic Rows

The final code provides the six canonical names in table 9.12, while retaining historical names as compatibility aliases: `compact` to `low`, `default` to `medium`, `high-recall` to `high`, `compact-with-image-input` to `low-with-image-input`, `default-with-image-input` to `medium-with-image-input`, and `visual-recall` to `medium-with-visual-recall`. The small-context run in table 9.5 is kept only as a lower-bound diagnostic, not as a supported preset. The historical zero-image row remains only to document that the original image flag produced no image URLs. It is not a valid image-input experiment.

A.8. V3 Retrieval-Backbone Archive

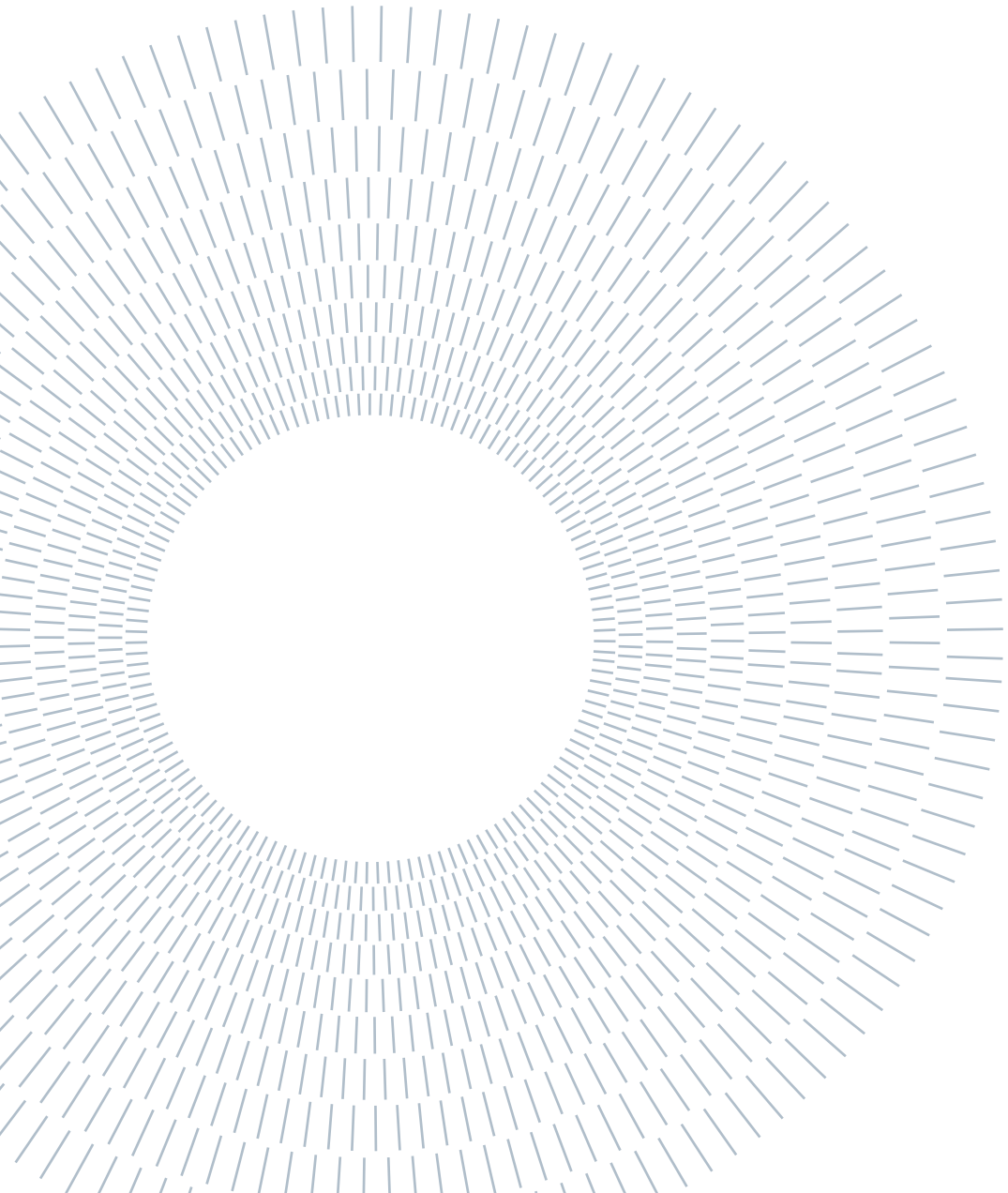
The v3 text-retrieval-backbone experiment is archived under `thesis-v3/experiments/retrieval-backbone-v3`. The summarized report is `thesis-v3/retrieval-backbone-results.md`. The raw archive keeps retrieval-only run JSON files, Qwen answer directories, Codex scoring CSVs, and derived summary tables for Phase 1, Phase 2, the RRF sweep, and the final 200-question confirmation.

The additional matched v2-high control is archived under `thesis-v3/experiments/matched-v2-high-qwen3-4b`. Its written addendum is `thesis-v3/matched-v2-high-qwen3-4b-results.md`. The archive keeps the generated 200-question query file, the Qwen3-4B high-recall retrieved contexts, Qwen answers, Codex-over-context answers, unified content-only Codex reviewer scores, and the final five-row comparison CSV.

Phase 1 row	Mean	High	Low	Avg ctx	Latency	Tokens	Index/DB
p1-dense-qwen3-4b	1.5000	0.280	0.720	7495.08	3.64	8300.46	1004.74s / 36.49MB
p1-sparse-bm25	1.3800	0.240	0.720	10244.32	4.42	10524.98	4.26s / 20.33MB
p1-dense-qwen3-0p6b	1.3600	0.260	0.740	7554.20	3.70	8400.56	39.97s / 18.94MB
p1-sparse-bge-m3	1.3600	0.240	0.740	10528.84	4.47	10984.74	33.88s / 21.60MB
p1-dense-mxbai	1.2800	0.220	0.760	7190.44	3.89	8315.40	991.53s / 18.94MB
p1-dense-bgelarge	1.2600	0.220	0.760	6100.56	3.41	7137.46	1143.11s / 18.94MB
p1-dense-qwen3-8b	1.2600	0.220	0.760	7379.58	3.61	8174.24	1059.36s / 54.63MB
p1-sparse-splade	1.2600	0.200	0.760	9658.00	4.74	10303.82	252.90s / 20.40MB
p1-dense-e5large	1.2000	0.200	0.780	9961.28	4.42	10562.36	613.35s / 18.94MB
p1-sparse-bm42	1.1000	0.180	0.780	10244.78	4.17	10658.68	46.49s / 19.45MB
p1-dense-arctic	0.9000	0.140	0.820	4885.30	2.98	6564.64	830.45s / 18.94MB

Table A.7: Complete v3 Phase 1 single-route screening table

Across the Qwen answer-generation runs created for v3, the archive contains 1250 generated answers, 12,211,348 Qwen prompt tokens, 147,150 Qwen completion tokens, and 12,358,498 total Qwen tokens. The summed Qwen answer-generation latency is 5377.3 seconds. Codex reviewer token usage is not included because the Codex CLI reviewer did not provide complete token accounting for the scoring calls.



Bibliography

[1] Z. Cheng. RAG-Flow: Multimodal retrieval-augmented generation pipeline for technical manuals. `https://github.com/ziyan-c/RAG-Flow`, 2026. Accessed 2026-05-29.

[2] G. V. Cormack, C. L. A. Clarke, and S. Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759, 2009. doi: 10.1145/1571941.1572114.

[3] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert. RAGAS: Automated evaluation of retrieval augmented generation. *arXiv preprint arXiv:2309.15217*, 2023. URL `https://arxiv.org/abs/2309.15217`.

[4] M. Faysse, H. Sibille, T. Wu, B. Omrani, G. Viaud, C. Hudelot, and P. Colombo. ColPali: Efficient document retrieval with vision language models. *arXiv preprint arXiv:2407.01449*, 2024. URL `https://arxiv.org/abs/2407.01449`.

[5] Y. Huang, T. Lv, L. Cui, Y. Lu, and F. Wei. LayoutLMv3: Pre-training for document AI with unified text and image masking. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 4083–4091, 2022. doi: 10.1145/3503161.3548112.

[6] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL `https://aclanthology.org/2020.emnlp-main.550`.

[7] O. Khattab and M. Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 39–48, 2020. doi: 10.1145/3397271.3401075.

[8] G. Kim, T. Hong, M. Yim, J. Nam, J. Park, J. Yim, W. Hwang, S. Yun, D. Han, and S. Park. OCR-free document understanding transformer. In *Computer Vision*

– *ECCV 2022*, volume 13688 of *Lecture Notes in Computer Science*, pages 498–517, 2022. doi: 10.1007/978-3-031-19815-1_29. URL https://www.ecva.net/papers/eccv_2022/papers_ECCV/html/8042_ECCV_2022_paper.php.

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W.-t. Yih, T. Rocktaschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

[10] MinerU Team. MinerU: Document parsing tool for machine-readable markdown and JSON. https://opendatalab.github.io/MinerU/, 2026. Accessed 2026-05-24.

[11] Qdrant. Qdrant: Vector search engine. https://qdrant.tech/, 2026. Accessed 2026-05-24.

[12] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

[13] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

[14] SGLang Team. SGLang: High-performance serving framework for large language models and multimodal models. https://github.com/sgl-project/sglang, 2026. Accessed 2026-05-24.