GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Bachelor's Thesis

submitted in partial fulfillment of the
requirements for the course "Applied Computer Science"

The Impact of Data Preparation Strategies on Retrieval Quality in Educational RAG Systems

Corinna Margarete Liley

MatrNr: 16317955

First Supervisor: Julian Kunkel
Second Supervisor: Jonathan Decker

Georg-August-Universität Göttingen
Institute of Computer Science
ISSN: 1612-6793

August 22, 2026

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 www.informatik.uni-goettingen.de

Abstract

Large Language Models (LLMs) are increasingly used as tutoring systems in university education, but they hallucinate on course-specific material that never appeared in their training data. Retrieval-Augmented Generation (RAG) addresses this by providing LLMs with external knowledge to ensure factual accuracy, but even a strong model retrieves poorly from unstructured data. Existing work focuses on optimising model performance, while treating the corpus as fixed and evaluates on public corpora whose queries are not comparable to those a deployed tutoring system actually receives. This thesis therefore begins by building a benchmark of 83 questions that students actually asked their tutors during a C programming course, with 342 graded gold references at chunk level. On this basis, holding the embedding model and hybrid retriever constant, we compare three stages of data preparation: naive fixed-size chunking (V1), structure-aware chunking with hierarchy embedding (V2), and keyword injection via KeyBERT (V3a) or an LLM (V3b). Preparation alone raises chunk-level nDCG from 0.412 to 0.546, and statistical keyword extraction fails on this corpus where LLM-generated keywords succeed, although that gain is confined to ranking and routing: the coverage metrics peak one stage earlier, at V2. None of this costs query time: local retrieval stays at 14–20 ms in every version, while end-to-end latency (mean 3.3–4.3 s) is dominated entirely by a single remote embedding call, so the real bottleneck lies outside the data pipeline. The evidence-depth metric defined for this benchmark stays flat which a granularity-matched control baseline identifies as an artefact of rank-based metrics on a growing index rather than a retrieval failure.

Statement on the Use of AI Tools

How was AI used when working with literature and sources?
I used asta.allen.ai to search for papers and publications. To better understand how the sources relate to each other, I also used NotebookLM. I checked all content by hand against the papers.

How was AI used in the process of academic writing?
When I had trouble writing something in English, I used Claude Sonnet to translate single sentences or to suggest wordings. Claude Sonnet also reviewed the language of the finished text.

How was AI used in the methodology, especially for collecting and analysing data?
For the data analysis and for making the plots in LaTeX, I used Claude Opus to write Python code and to create the LaTeX plots. I reviewed the results. For the benchmark, I used NotebookLM to search for evidence passages in the course materials. I did this step by step: I reviewed the passages that the AI found, then changed and extended the search when needed, until the results met my requirements.

How was AI used in other activities?
For programming, I worked with Claude Code Opus step by step: first the AI wrote an implementation plan, which I reviewed and revised. Then the AI wrote the code, and I reviewed the result again. I repeated this cycle until the code met my requirements.

Contents

List of Tables v

List of Figures vii

List of Listings ix

List of Abbreviations x

1 Introduction 1

2 Theoretical Background 5
2.1 RAG . . . 5
2.2 Information Retrieval . . . 6
2.3 Chunking . . . 6
2.4 Data-Centric AI . . . 7
2.5 Evaluation Metrics . . . 8

3 Related Work 9
3.1 RAG Evolution . . . 9
3.2 Strategies to Optimise Retrieval Quality . . . 10
3.2.1 Hybrid Retrieval . . . 10
3.2.2 Chunking Strategies . . . 10
3.2.3 Keyword and Metadata Augmentation . . . 11
3.3 RAG-Based AI Tutoring Systems . . . 12

4 Methodology 13
4.1 Architecture . . . 13
4.2 Data Sources . . . 15
4.3 Evolution Stages of the Data Ingest . . . 15
4.3.1 V1: Naive Baseline . . . 15
4.3.2 V2: Structural Chunking with Hierarchy Embedding . . . 16
4.3.3 V1-215: Granularity Control Baseline . . . 17
4.3.4 V3: Keyword Injection . . . 18
4.4 Comparative Analysis: With and Without CTC . . . 19
4.5 Gold Standard Benchmark Construction . . . 19
4.5.1 Source of Queries . . . 19
4.5.2 Annotation Process . . . 20
4.5.3 Annotation Structure . . . 20
4.5.4 Evidence-Coverage Validation . . . 21
4.6 Evaluation Metrics . . . 22
4.6.1 Score-Weighted Chunk-Level Metrics . . . 23
4.6.2 Out-of-Scope Handling (AUROC) . . . 25

5 Results **26**
5.1 Impact of Iterative Preparation 27
5.2 Effects of Structural Chunking 28
5.3 Statistical vs. Generative Keyword Injection 31
5.4 Impact of filling informational gaps 32
5.5 Impact of Query Type 33
5.6 Influence of Source-Document Types 34
5.7 Detection of Unanswerable Queries 35
5.8 Quality vs. latency 36

6 Discussion **37**
6.1 The flat WRS paradox 37
6.2 The Multi-Hop Bottleneck 39
6.3 Why Statistical Extraction Fails 40
6.4 Implications for Deployment 41
6.5 Limitations 43

7 Conclusion **46**

References **48**

A Code Listings **A1**

B Empirical Validation of WRS **A4**

C WRS and Annotation Density **A5**

D Keyword Quality Inspection **A6**

E LLM Prompt for V3b **A8**

F Full Retrieval Metrics **A9**

List of Tables

1 Chunk statistics per ingest version, measured on the cached chunk files of the full corpus (including the CTC). Word counts are whitespace-separated tokens, the unit the chunker operates on. The corpus-wide median of 215 words that motivates the granularity control (Section 4.3.3) is a mixture of two very different source types: V2's PDF chunks are considerably larger and its YAML chunks considerably smaller than that single figure suggests. Excluding the CTC, which is the primary evaluation condition (Section 4.4), the indexes contain 254 (V1), 453 (V1-215) and 432 chunks (V2, V3a, V3b). 18

2 Evaluation metrics and the question they answer. All chunk-level @k metrics are evaluated at k=20. Chunk-MRR and Chain-MRR are rank-based but computed within the same top-20 list. 26

3 Core metric overview (without Curated Tutoring Collection (CTC)). The columns do not share one sample: Hit@1, Normalised Discounted Cumulative Gain (nDCG), Weighted Relevance Score (WRS) and WRec are averaged over all $n = 72$ in-scope queries, Recall@20 and Strict@20 over the $n_{\text{appl}} = 66$ of them that carry at least one G_{high} reference, and Chain-MRR over the $n_{\text{appl}} = 50$ complementary queries for which it is defined. All values are ratios in $[0, 1]$; Strict@20 is the share of queries in which every G_{high} reference is retrieved. Best value per column in bold. The full table including Mean Reciprocal Rank (MRR) and Hit@5, and the V1-215 control, is given in Table 7 in the appendix. 27

4 Correlation between the number of gold references per query and two chunk-level metrics ($n = 72$ in-scope queries, without CTC). WRS is consistently and negatively correlated with the reference count, Chunk-MRR is not. A5

5 Keyword sets injected in V3a and V3b, measured on the cached chunk files (451 chunks in each version). Counts are per extracted phrase, not per comma-separated token: four V3a phrases contain literal commas and would otherwise be counted more than once. "Markup or code characters" counts keywords containing at least one of `# { } ( ) < > | " ; %` or consisting solely of a C syntax token. The five most frequent V3a keywords are `# kapitel` (104 occurrences), `c-code` (87), `compiler` (46), `# include < stdio.h >` (41) and `# anhang` (32); for V3b they are `dynamischer Speicher` (108), `Heap` (63), `int` (60), `malloc` (59) and `printf` (57). A6

6 Injected keyword headers for three chunks, V3a (KeyBERT) against V3b (LLM), reproduced verbatim from the cached chunk files including the conversion artefacts that KeyBERT selects. Long literals are truncated with … where marked. A6

7 Full retrieval metrics by version and CTC condition. All @k metrics are evaluated at the cut-offs given in the header; all values are ratios in $[0, 1]$, Hit@k and Strict@20 being shares of queries. n = evaluated queries; the with-CTC rows average over all 83 queries and therefore include the eleven CTC-only queries that are out of scope in the without-CTC rows. Chain-MRR* is defined only on complementary queries ($n = 54$ with CTC, $n = 50$ without); Recall@20 and Strict@20 on the queries carrying at least one G_{high} reference ($n = 83$ with CTC, $n = 66$ without). A9

8 Out-of-scope separability (AUROC) and latency components by version and CTC condition. AUROC is defined only without the CTC, since with it all 83 queries are in scope (the 11 CTC-only queries become out-of-scope only when the collection is removed). “Infeas.” counts queries whose gold evidence no chunk of the index contains. Latencies in milliseconds. A9

List of Figures

1 Five-phase evaluation pipeline. The three data preparation stages (V1–V3b) are only distinct in Phase 1; all downstream phases share a fixed retrieval and evaluation infrastructure. Orange dashed arrows indicate the caching to disk: repeated runs skip these steps and load from cache. The corpus-hash (`.sha256`) ensures that a change in chunk texts triggers a full embedding rebuild. 14

2 Hard-negative scenario from the evaluation corpus. The retrieval system returns a chunk about pointer syntax from the correct source document (*programmierung_mit_ansi_c-127-148.pdf*, 23 chunks), but not the chunk containing the theoretical definition of dereferencing. A document-level metric counts this as a success (false positive), chunk-level evaluation correctly identifies it as a miss. 23

3 Document- and chunk-level metrics across the four data-preparation versions (without CTC, $n = 72$). Hit@1 is document-level; nDCG, WRS, and Chain-MRR are chunk-level. Hit@1 and nDCG rise monotonically while WRS stays flat, the paradox examined in RQ2–RQ3. 28

4 Control for granularity (without CTC, $n = 72$; levels, so the 17 queries whose evidence a 215-word window cuts apart are included at the score they achieve, Section 4.5). V1-215 applies V1's structure-blind fixed-size splitting at V2's median chunk length (215 words) to separate chunk size from the rest of the V2 preparation. It underperforms *both* configurations on all metrics, indicating that the V2 gain does not come from the smaller chunks; part of V1-215's poor performance is lost evidence rather than a ranking effect (Section 6.5). 29

5 Paired step-gains from V1 to V2 (without CTC, $n = 68$ after excluding the four queries whose evidence is infeasible in V1; Chain-MRR on the $n = 62$ of these that carry at least one G_{high} reference). Filled markers indicate significance (95 % bootstrap Confidence Interval (CI) excludes 0). Structural chunking significantly improves ranking quality (nDCG) but deepens the multi-hop bottleneck (Chain-MRR). 30

6 Rank-Cumulative Distribution Function (CDF) of found gold references (all in-scope queries, without CTC). V2 retrieves more evidence overall (63.4 % → 65.9 %) but at higher (worse) ranks, surpassing V1 only from rank 13 on. This is why Weighted Recall (WRec) rises while WRS does not: the two metrics share the same weighting and differ only in the rank-decay term (Section 4.6.1), so their divergence isolates *when* evidence is retrieved from *whether* it is retrieved. 30

7 Paired step-gains from V2 to V3a and V2 to V3b for all metrics (without CTC, $n = 72$; Chain-MRR on the $n = 66$ queries that carry at least one G_{high} reference). V3a produces no significant gain. V3b significantly improves Hit@1 and Chain-MRR (the latter with a CI close to zero) while leaving WRS unchanged. Filled markers = significant (95 % bootstrap CI excludes 0), open = n.s. 31

8 Impact of adding the Curated Tutoring Collection to the source data.
(A) *How does the CTC affect out-of-scope queries ($n = 11$)?* Without the CTC, WRS is 0. With it, mean WRS reaches 0.78 (V1) to 0.91 (V2–V3b). V2+ benefits from splitting the CTC into 19 structural articles.
(B) *How does the CTC affect already answerable queries ($n = 72$)?* The additional source provides a small, version-independent gain across all versions (open markers = without CTC, filled = with CTC. * = significant, 95 % bootstrap CI excludes 0). Both conditions are averaged over the same 72 queries, so the with-CTC values here are not the $n = 83$ ones of Table 7, which also average in the eleven CTC-only queries of panel (A). 32
9 Impact by query type (without CTC).
(A) Chunk-MRR for redundant ($n = 18$) vs. complementary ($n = 54$) queries: redundant queries rise consistently at every stage ($0.410 \rightarrow 0.486$), complementary queries stay similar ($0.627 \rightarrow 0.661$, including a small V3a KeyBERT dip from RQ3).
(B) Chain-MRR on the $n_{\text{appl}} = 50$ complementary queries for which it is defined, captures the V1→V2 drop because of evidence distribution (RQ2) and its recovery with LLM keywords (V3b). 33
10 V1→V2 paired Δ by source-document type (feasible queries, without CTC). Left: WRS@20 (the RQ2 fragmentation cost); right: nDCG@20 (the RQ2 ranking-quality benefit). 95 %-bootstrap CIs, filled marker and * = significant (CI excludes 0). nDCG rises for all three types, WRS falls most for YAML exercises, which V2 splits into separate task and solution chunks, though no WRS interval excludes zero. The benefit of the structural preparation is universal, its dispersion cost is not. 34
11 Area under the Receiver Operating Characteristic (ROC) Curve (AUROC) for separating in-scope ($n = 72$) from out-of-scope ($n = 11$) queries by best retrieval score, per version (without CTC). All curves are far above the chance diagonal. AUROC: V1 = 0.907, V2 = 0.924, V3a = 0.909, V3b = 0.928 (best of the four pipeline versions). A higher curve means unanswerable queries can be rejected at a lower cost in wrongly rejected answerable queries. 35
12 Importance vs. retrieval rank for every found gold reference (without CTC, V1 and V2). The y-axis is reversed so higher means retrieved earlier; points are jittered to separate the discrete score and rank values. High-importance references (right) concentrate at top ranks in both versions—$G_{\geq 0.8}$ evidence is found at median rank 3 (V1) / 4 (V2) versus 6 / 5 for lower-weighted references—confirming that the retriever prioritises must-have evidence; the slightly deeper V2 cloud reflects the evidence dispersion analysed in RQ2 (Section 5.2). A4

Listings

1 V2/V3 structural chunking (`Code/loaders.py`). Splits at heading boundaries, keeps code fences atomic, and applies a 20 % sliding overlap between neighbouring chunks. `MIN_TOKENS` only forces a flush *within* a section; a new heading (`force=True`) always starts a new chunk regardless of size, which is why the resulting chunk sizes vary so widely (Table 1). A1

2 WRS@K computation for a single query (`Code/validation.py`). References without an evidence snippet are skipped entirely (line 15); references whose evidence is not found in the top-k contribute their `matching_score` to the denominator but nothing to the numerator (line 21), which is what makes WRS sensitive to how many references a query carries (Section 6.1, Appendix C). `_has_evidence` implements the exact-substring/LCS matching described in Section 4.6 and is omitted here. A2

3 Chain-MRR computation (`Code/reporting.py`). Returns `None` when a query has no G_{high} reference (not applicable, excluded from the mean), `0.0` when at least one G_{high} reference is missing from the top-k (applicable, failed), and otherwise the reciprocal of the *worst* (highest) rank among the G_{high} references – the bottleneck that motivates reporting Chain-MRR alongside Chunk-MRR (Section 4.6.1). A3

List of Abbreviations

API Application Programming Interface

BM25 Best Matching 25

IR Information Retrieval

LLM Large Language Model

MMR Maximal Marginal Relevance

MRR Mean Reciprocal Rank

nDCG Normalised Discounted Cumulative Gain

OCR Optical Character Recognition

PDF Portable Document Format

RAG Retrieval-Augmented Generation

RRF Reciprocal Rank Fusion

DCAI Data-Centric AI

CTC Curated Tutoring Collection

ROC Receiver Operating Characteristic

AUROC Area under the ROC Curve

CDF Cumulative Distribution Function

WRS Weighted Relevance Score

OOS Out-of-Scope

WRec Weighted Recall

CI Confidence Interval

POS Part-of-Speech

HNSW Hierarchical Navigable Small World

1 Introduction

In recent years, Large Language Models (LLMs) have made a profound impact in almost every domain, leading to their rapid integration into education as AI-driven tutoring systems [26]. Especially in domains such as programming classes, LLMs could serve as interactive learning partners, assisting students with debugging or with understanding complex concepts. In practice, however, these models tend to produce factually incorrect answers and hallucinations in specialised domains [3, 28].

The main causes of this behaviour are static training data and "long-tail" knowledge. First, a model can only work with what was present in its training data up to a fixed cutoff: it has no access to real-time information or to domain-specific material published afterwards [3, 7, 20]. In university education, this means that specific course rules, current exercise sheets, or internal examination conventions remain inaccessible to the model. Second, even state-of-the-art models struggle to internalise information that appears only rarely in their training corpus, the so-called "long-tail" knowledge [21]. Since course-specific information and exercise material are not included in public training corpora, models tend to fill these gaps with generic, factually incorrect statements [28, 32]. However, since the correctness and precision of information in programming education is essential, the unreliability of these models becomes a barrier to deployment in university education.

To address hallucination, the architecture of *Retrieval-Augmented Generation (RAG)* has become more and more established [3, 25]. Instead of relying only on internal memory, the LLM is provided with external documents that are retrieved from up-to-date knowledge bases before the query is answered [11]. Although RAG systems reduce the hallucination rate, their success depends on the quality of the underlying data. Poorly prepared data can lead to "garbage in, garbage out" scenarios, where the retriever fails to find relevant information [18, 24].

Existing approaches frequently focus on the optimisation of model architectures and hyperparameters while keeping the dataset fixed [17, 24]. In our setting, the University of Göttingen is preparing an AI tutoring assistant for the C programming course within the *GAIA* project.[1] The retrieval layer such a system builds on is *Arcana*, the RAG service of the GWDG.[2] Arcana has so far been tried out for this course internally rather than deployed to students, because both stages it consists of are naive and unoptimised: the ingest pipeline segments documents without considering their structure and the retrieval stage restricts each search to a single group of the index rather than searching the corpus as a whole (Section 4.1). The source material (lecture scripts converted from PDF, structured YAML exercise files, and a hand-curated tutoring collection we introduce) is heterogeneous enough that the *way* documents are split and annotated directly determines what the retriever can find at all. Which ingest configuration makes this material retrievable is therefore a question that has to be answered before a tutoring system can be built on top of it.

Because Arcana in its current form cannot be reconfigured in the way such an experiment requires, this thesis takes *zVec* [1] as its basis: an open-source in-process vector database developed by Alibaba Group. Like Arcana, it answers a query by vector similarity over an index of embedded chunks, but unlike Arcana it leaves the ingest configurable and searches

[1] *Generative Artificial Intelligence Agent*, a project of the University of Göttingen developing an AI-based personal tutor for students: `https://www.uni-goettingen.de/de/706531.html`.

[2] `https://docs.hpc.gwdg.de/services/ai-services/arcana/index.html`

that index as a whole rather than one group of it. It is therefore structurally close enough to Arcana to serve as a model for a more optimised version of it, and every configuration evaluated in this thesis is one that an optimised Arcana ingest could adopt (Section 4.1).

This work is part of ongoing optimisation work on the RAG systems used at the University of Göttingen and the GWDG. It builds in particular on Kochanke [23], whose evaluation of over 600 chunking configurations establishes chunking as the primary control variable for retrieval quality and supplies several of the parameter choices adopted here (Sections 2.3 and 4.3). This work therefore follows a **Data-Centric AI** approach, where the model and retrieval architecture are held constant across all experiments and only the data preparation varies [42, 17].

This work surveys three stages of data preparation: a naive fixed-size chunking baseline (V1), structural chunking that considers document hierarchies (V2) and keyword injection to fill the vocabulary gap between short student queries and long lecture notes (V3). Retrieval is configured as a hybrid dense + sparse method [35, 9] across all stages, so that the variation of the data preparation can be measured in isolation.

The goal of this work is the systematic investigation of how data-centric preparation strategies affect retrieval quality in a C programming tutoring domain. To make this measurement possible in the first place, the work constructs a manually annotated benchmark of real student queries with chunk-level gold references which is used as a reference dataset that permits evaluation at the granular level of individual passages rather than whole documents. This naturally leads to the main research question:

> *How do data-centric preparation strategies impact retrieval quality in a RAG-based tutoring system for a university C programming course?*

We decompose this question into eight sub-questions.

RQ1 Does an iterative data-preparation approach consistently improve retrieval quality?

RQ2 Does structural chunking help and does this fragmentation make it harder to find evidence?

RQ3 Does keyword injection help, and does it matter whether keywords are extracted statistically (KeyBERT) or generated by an LLM?

RQ4 Does the hand-curated tutoring collection improve performance and does it benefit each version differently?

RQ5 Does the benefit depend on query type (redundant vs. complementary)?

RQ6 Are the improvements consistent across source-document types (YAML exercises vs. PDF slides)?

RQ7 Can the system distinguish answerable from unanswerable queries based on its retrieval scores alone?

RQ8 What is the trade-off between retrieval quality and query latency across the versions?

Evaluation uses established ranking metrics (Hit@k, MRR, nDCG [29, 19, 40]), but unlike most IR benchmarks that operate at the document level, this thesis shifts the evaluation focus to the chunk level: for each query, gold references point to specific text passages rather than whole documents. This matters because retrieving the correct

document does not guarantee the retrieval of the precise passage required for an accurate answer. Therefore, this work uses score-weighted chunk-level metrics for evaluating the graded expert annotation of the benchmark and it demonstrates why document-level metrics can lead to misleading conclusions.

The results show that naive chunking (V1) suffers from poor ranking quality (chunk-level nDCG = 0.412). Structural chunking (V2) raises nDCG to 0.526, showing that context fragmentation is an essential reason for failure in fixed-size chunking. Among keyword injection strategies, LLM-generated keywords (V3b) consistently outperform the statistical KeyBERT injection (V3a), achieving the best Hit@1 overall (+8.3 pp over V2) and recovering the multi-hop bottleneck that structural chunking had opened. At the same time, the metric we introduce to capture how early the weighted evidence is actually read (WRS) remains mostly unchanged across all versions (0.303 → 0.300). A granularity-matched control baseline (V1-215), which applies the naive splitting at V2's chunk size and therefore at V2's index size, shows this to be mostly attributable to the metric rather than retrieval failure: once index size is held constant, WRS improves along with every other chunk-level metric. A comparative analysis shows that the Curated Tutoring Collection we introduce widens *coverage*: eleven previously unanswerable queries become answerable (mean WRS 0.78–0.91), while gains on already answerable queries are small and version-independent. Testing against a small set of 11 Out-of-Scope (OOS) queries achieves an AUROC $\geq$ 0.907 across all four pipeline versions (best: V3b with 0.928), suggesting that retrieval scores alone are a usable signal for rejecting unanswerable queries. Query latency, finally, is unaffected by data preparation: local retrieval takes 14–20 ms in every version, and end-to-end time (mean 3.3–4.3 s) is determined almost entirely by a single remote embedding call.

Contributions. This thesis makes three contributions. First, it constructs a benchmark that makes a representative evaluation of this system possible in the first place. Retrieval components are commonly evaluated on public, general-purpose corpora whose queries are different from the ones a deployed system actually receives, so methods that perform well on such corpora do not reliably transfer to a specific domain [39]. For this reason, domain-specific benchmarks are needed [37, 41], which we constructed here. These references are placed at *chunk* level, so they point to the specific passage required for an answer rather than to a whole document, which introduces the second contribution. The benchmark consists of 83 questions that students actually asked their tutors during the C programming course, together with 342 gold references annotated by the author and validated by the course instructor [3]

Second, this work introduces a chunk-level evaluation methodology, that exploits the graded expert annotations through score-weighted ranking metrics, making it possible to distinguish *whether* the required evidence is retrieved from *how early* it is retrieved. This distinction cannot be measured by document-level metrics.

Third, it provides a data-centric evaluation (DCAI) of three chunking and annotation strategies for RAG in a C programming course, isolating the effect of data preparation under a fixed embedding model and retrieval architecture. The central findings of this evaluation are invisible at document level and would not have been measurable without the first two contributions.

[3]The benchmark, the author-written Curated Tutoring Collection, and the pipeline code are available at `https://gitlab.gwdg.de/corinna.liley/c-data-centric-ai-thesis`; see Section 4.5.

Scope. The subject of this work is the *retrieval component* of a RAG system. The three strategies we evaluate (naive chunking, structure-aware chunking and keyword injection) are some of the most established data-preparation techniques in RAG [23, 37]. Techniques outside this scope, such as reranking, query expansion, fine-tuned embedding models, or graph-based retrieval, are not evaluated. Moreover, the generation component, end-to-end student experience, and multi-turn conversation management are not examined.

Structure. This thesis has the following structure. Section 2 introduces the theoretical background covering RAG, Information Retrieval, chunking, and the evaluation metrics used in this work. Section 3 reports related work on the evolution of RAG, strategies of retrieval, chunking and keyword augmentation, and RAG-based tutoring systems. Section 4 describes the system architecture, the three data preparation stages, a comparative analysis of how supplementary information impacts retrieval quality, the benchmark construction and the chunk-level evaluation metrics that are used. The answers to RQ1–RQ8 are reported in Section 5 while Section 6 interprets the findings and points out implications that can be used for the Arcana deployment. Section 7 summarises the contributions and outlines directions for future work.

2 Theoretical Background

This chapter explains the foundations on which we build the experiments. Section 2.1 introduces the Retrieval-Augmented Generation architecture and defines its retrieval phase as the object of this study. Section 2.2 then describes the information-retrieval principles behind the hybrid retriever used throughout this work. Section 2.3 motivates chunking as the first central control variable of retrieval quality, Section 2.4 presents the experimental design within the Data-Centric AI paradigm, and Section 2.5 defines the ranking and efficiency metrics on which the evaluation relies.

2.1 RAG

The quality of language models as knowledge sources depends on their **parametric knowledge**, the information they are trained on [25]. Lewis et al. describe these models as functioning as a "parameterized implicit knowledge base": all knowledge is static, fixed at training time, and cannot be revised or expanded without re-training [25]. Once built, a model's knowledge base cannot evolve [7], so queries that depend on domain-specific or post-training information cannot be answered reliably from parametric memory alone [20]. A common solution is to complement parametric memory with **non-parametric** (retrieval-based) memory, an external knowledge base that can be updated, inspected, and queried after training time [25]. Lewis et al. describe this as the difference between a *closed-book* model, one that relies entirely on memorised knowledge, and an *open-book* model that has access to external documents at inference time [25, 28].

The RAG architecture combines the advantages of generative models with the precision of dedicated retrieval systems [25]. Its core framework is indexing, retrieval and generation [11]; at query time we distinguish three phases:

1. **Retrieval:** The retriever searches the database for the top-k most relevant chunks.
2. **Augmentation:** These chunks are embedded together with the original user query in a prompt template.
3. **Generation:** The LLM generates an answer based on the provided context. Because the answer is grounded in retrieved documents rather than parametric memory alone, the probability of hallucinations is reduced [3, 28].

As defined in the introduction, the subject of this work is only the first phase. Augmentation and generation are named for completeness of the theoretical framing but are not measured or optimised. The quality of the first phase is, however, a necessary condition for the quality of the following phases [37]: without correct retrieval, even a high-quality generator cannot produce a correct answer. The mechanics of this retrieval phase (dense, sparse, and hybrid retrieval) are the subject of the following section.

2.2 Information Retrieval

Within the retrieval phase defined above, the goal of Information Retrieval (IR) is the identification and extraction of the most relevant document segments from a knowledge base for a given user query [29, Ch. 1].

Modern IR systems use **Dense Retrieval** to build a bridge between the vocabulary of user queries and the indexed text. Chunks are transformed into high-dimensional vectors (*embeddings*) by neural networks, creating a shared vector space so that semantically related texts are geometrically close even when they do not share any keywords [22]. The mathematical similarity between a query q and a document d is computed via **cosine similarity**:

$$\text{sim}(q, d) = \frac{q \cdot d}{\|q\|\|d\|} \tag{1}$$

The specific embedding model and index configuration used in this work are described in Section 4.1. While dense retrieval methods capture semantic similarity, lexical methods retrieve only documents that contain the query terms themselves [39], which suits exact matches such as code syntax. **Hybrid retrieval** architectures therefore combine both approaches: a *sparse* component such as Best Matching 25 (BM25) [11, 22] indexes token frequencies and identifies exact matches, while a *dense* component captures semantic proximity. Both components each produce a ranking of the corpus documents, which must then be combined.

The **Reciprocal Rank Fusion (RRF)** [9] used in this work is a low-parameter fusion method that aggregates the final position of a document from the reciprocal ranks across all input lists:

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + \text{rank}_r(d)} \tag{2}$$

Here R is the set of input rankings and k is a smoothing constant (in this work $k = 60$, the value proposed by Cormack et al. [9]). RRF requires no score calibration between components and is robust to differences in score scales.

2.3 Chunking

Before being indexed, source documents are pre-processed into **chunks**, related text segments of manageable size. The granularity and quality of this segmentation directly affects retrieval performance: if the size is too large, a chunk contains irrelevant context when searching for information; if it is too fine, related information is fragmented across several chunks [38, 37]. This *context fragmentation* problem is the primary motivation for the structural chunking stage evaluated in this work. Kochanke [23], evaluating over 600 chunking configurations in a RAG system, establishes chunking as the primary control variable for retrieval quality. While the embedding model mainly determines technical latency, the chunking configuration determines which evidence is available to the language model at all.

2.4 Data-Centric AI

The design of machine learning systems has traditionally been *model-centric*: given a fixed dataset, the developer iterates on the model architecture, its hyperparameters, and its training procedure to improve performance. Data-Centric AI (DCAI) changes this perspective. Jakubik et al. [17] give formal definitions for both paradigms:

> *"Model-centric artificial intelligence is the paradigm focusing on the choice of the suitable model type, architecture, and hyperparameters from a wide range of possibilities for building effective and efficient AI-based systems."*

> *"Data-centric artificial intelligence is the paradigm emphasizing that the systematic design and engineering of data are essential for building effective and efficient AI-based systems."*

In the DCAI paradigm, the model and its training procedure are held constant while the data pipeline is iterated: data is inspected, errors are identified, the representation is refined, and the system is re-evaluated. Zha et al. [42] organise the field around three general data-centric goals: training data development, inference data development, and data maintenance. Especially in specialised domains with constrained resources, improving data quality can lead to better results than scaling or fine-tuning the model [42, 17].

This thesis applies the DCAI paradigm to the retrieval component of a RAG system. The model (embedding model, retrieval algorithm, and index) is held constant across all experiments. The varied element is only the data ingest pipeline: the strategy by which source documents are segmented, represented, and enriched before indexing. The three ingest stages V1, V2, and V3 (consisting of V3a and V3b) represent three iterations of this data engineering loop, applied to the same fixed model infrastructure.

2.5 Evaluation Metrics

Quality Metrics To evaluate the quality of ranked search results, the following standard IR metrics are used:

- **Hit@k** measures whether at least one relevant document appears in the top-k retrieved results (reported as Recall@k in [30]). Computed for $k \in \{1, 5\}$; Hit@1 corresponds to Precision@1 (exact match). Operates on deduplicated document IDs: when multiple chunks from the same document appear in the top-k, only the first occurrence is counted.

- **MRR:** Evaluates the position of the first relevant hit across a set of queries Q [32, 29]:

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i} \tag{3}$$

  where rank_i is the rank of the first relevant result for query i, and queries for which no relevant result is found within the top-k contribute zero.

- **Normalised Discounted Cumulative Gain (nDCG@k):**

$$\mathrm{DCG}[i] = \begin{cases} G[1] & \text{if } i = 1 \\ \mathrm{DCG}[i-1] + G[i] \,/\, \log_2 i & \text{otherwise} \end{cases} \tag{4}$$

  where $G[i]$ is the relevance grade at rank position i and the logarithm base is $b = 2$, i.e. the discount $\log_2 i$ [19]

$$\mathrm{nDCG@}k = \frac{\mathrm{DCG@}k}{\mathrm{IDCG@}k} \tag{5}$$

  IDCG@k is the ideal DCG, i.e. the value achieved by a perfectly ranked result list, in this work computed at chunk level (Section 4.6).

Score-weighted chunk-level metrics extending these standard measures and exploiting the graded expert annotation of the benchmark are defined in Section 4.6.

Efficiency Metrics Beyond accuracy, latency is critical for practical deployment in real-time teaching interaction [30]. We compute three components of retrieval latency:

- **Embedding API latency** (`embed_latency_ms`): time for the remote embedding call.

- **Local retrieval latency** (`clean_latency_ms`): time for cosine similarity computation, BM25 scoring, and RRF fusion.

- **End-to-end wall-clock time** (`e2e_latency_ms`): total elapsed time from query submission to ranked result list.

Mean and 95th percentile are reported for each component.

3 Related Work

This chapter explores the current state of research in three areas central to this thesis. Section 3.1 describes the evolution of RAG architectures and shows that they treat data preparation as fixed. Section 3.2 presents concrete strategies for improving retrieval quality: hybrid retrieval (Section 3.2.1), chunking strategies (Section 3.2.2), and keyword and metadata augmentation (Section 3.2.3). Section 3.3 gives an insight into domain-specific RAG deployments in education. Across all these areas, data preparation is the dominant bottleneck: Kumar et al. [24] note that in practical AI applications 80% of activities involve data preparation, while Jarrahi et al. [18] argue that approaching DCAI, the role of data becomes the central focus throughout the entire AI life cycle.

3.1 RAG Evolution

Since the introduction of RAG, the architecture has been expanded. Gao et al. [11] describe three paradigms: *Naive RAG*, where the core framework consists of indexing, retrieval, and generation; *Advanced RAG*, which focuses on pre-retrieval processing (e.g. query rewriting) and post-retrieval processing (e.g. reranking) to avoid query misunderstanding and retrieval redundancy of Naive RAG; and *Modular RAG*, which decomposes the pipeline into independent, reconfigurable modules with non-linear execution. Across all three, however, the ingested corpus is treated as a fixed input rather than as a variable to optimise.

Furthermore, Jeong et al. [20] and Asai et al. [4] address query-time adaptation. Jeong et al. suggest Adaptive-RAG, which trains a classifier to dynamically select the most suitable retrieval strategy based on estimated query complexity. That way, unnecessary retrieval overhead for simple queries is avoided. Asai et al. [4] introduce Self-RAG, which trains a language model to send out special *reflection tokens* (categorised into retrieval tokens and critique tokens) that control whether retrieval is needed, whether retrieved passages are relevant and whether the generated output is based on them. Both approaches work on a fixed indexed corpus and they treat data preparation as a pre-condition and not a variable. The data-centric approach pursued here acts at exactly the stage they treat as fixed.

3.2 Strategies to Optimise Retrieval Quality

3.2.1 Hybrid Retrieval

There is no specific retrieval method that dominates across domains. Thakur et al. [39] evaluate 10 retrieval systems on 18 datasets from diverse tasks and domains and find that BM25 is a robust baseline; dense and sparse retrieval models are computationally more efficient but often underperform other approaches in zero-shot settings. Their combination (sparse for exact matches, dense for semantic proximity; Section 2.2) is what hybrid systems exploit.

Sawarkar et al. [35] demonstrate that hybrid retrieval (blending dense and BM25 retrieval) reaches better precision compared to conventional keyword-based or vector-based retrieval alone, achieving 88.77% retrieval accuracy on the NQ dataset and an nDCG@10 of 0.87 on TREC-COVID (an 8.2% improvement over the benchmark). Cormack et al. [9] establish RRF as a fusion method that outperforms any individual system and the standard Condorcet Fuse method, by 4–5% on average (MAP) across multiple TREC experiments. Because of this evidence base, all pipeline versions in this thesis use Dense + BM25 via RRF.

3.2.2 Chunking Strategies

The segmentation of source text and documents into indexable chunks is one of the earliest steps in the RAG pipeline. Setty et al. [37] study the constraints of RAG pipelines for financial documents and explore strategies including different chunking techniques, query expansion, metadata annotations, re-ranking and embedding fine-tuning to improve the retrieval quality. They observe that when all chunks are treated equally, important information is lost, especially in documents with structured texts such as documents with headings and tables.

Yepes et al. [41] study chunking specifically for SEC financial reports and suggest element-based chunking that does not split at paragraph-level, but instead by structural elements (titles, texts, tables). Their results show that element-based chunking achieves the best question-answering accuracy and the highest retrieval quality while using only half the number of chunks compared to methods that ignore document structure (62,529 vs. 112,155 chunks), simultaneously reducing indexing cost and query latency. The fact that a purely structural change to the chunking step reaches the largest improvement in their pipeline provides strong evidence for treating chunking as a control variable rather than a fixed preprocessing detail.

Singh et al. [38] propose two stages of chunking: semantic chunking and chunk-level filtering. In the first stage, documents are segmented into semantically coherent, variable-length sections by grouping sentences based on cosine similarity. A new chunk is only created when the similarity between two sentences drops below a threshold of 0.8, indicating a shift in the subtopic. In the second stage, an LLM evaluates the semantic relevance of each retrieved chunk against the user query and filters out irrelevant chunks before generation. The authors observe that existing methods operate at the document level and fail to isolate relevant passages, a granularity problem that motivates chunk-level evaluation in this thesis (Section 4.6).

Ciancone et al. [8] introduce ChunkNorris, a heuristic-based technique for PDF parsing and chunking that does not rely on machine learning. Their benchmark evaluates parsing and chunking methods on execution time, energy consumption, and retrieval accuracy.

They conclude that ChunkNorris outperforms both baseline and more advanced techniques, demonstrating that heuristic methods are a practical alternative for resource-constrained RAG systems.

For heterogeneous document types, Allu et al. [2] show that tabular content in PDFs requires the linking of column headers with corresponding cell values, because standard RAG retrieval fails on queries requiring information from complex tables when cells are not linked to their headers. To address this issue, Auer et al. [5] present Docling, an open-source PDF conversion tool powered by specialised AI models for layout analysis (DocLayNet) and table structure recognition (TableFormer) that converts PDFs to JSON or Markdown while preserving reading order, figures and tables, running on commodity hardware. Pérez and Vizcaíno [31] describe an advanced ingestion process using LLM-powered Optical Character Recognition (OCR). It is a node-based extraction technique creating relationships between different types of information and generating context-aware metadata. They demonstrate improvements in answer relevancy and information faithfulness across multiple knowledge bases.

In conclusion, these works find that preserving document structure during ingestion (using headings, sections, and table boundaries) is essential for heterogeneous corpora, yet they treat it as a one-off engineering or tooling problem rather than as a controlled experimental variable. This motivates the structural chunking stage (V2, Section 4.3) evaluated in this thesis, which splits the PDF lecture material along its heading hierarchy and isolates the resulting retrieval effect under otherwise fixed model and retrieval conditions.

3.2.3 Keyword and Metadata Augmentation

A challenge in domain-specific retrieval, especially in education, is the vocabulary mismatch between short user queries and long structured documents, because terms of a user query may not appear in the source material (the *lexical gap* [39]). One strategy of metadata injection is to enrich chunks with additional terms at indexing time.

KeyBERT [12] builds n-gram embeddings of document and keyword candidates via a pre-trained BERT model and selects the candidates whose embeddings have the highest cosine similarity to the document embedding. This unsupervised approach requires no labelled training data and produces keywords grounded in the document's semantic content. The standard n-gram candidate generation is improved by Schopf et al. [36], who replace n-gram enumeration with a Part-of-Speech (POS) filter that admits only grammatical noun phrases as candidates. This design is motivated by the finding of Hulth [15] that noun phrases lead to significantly more meaningful keywords than only statistical candidates. To prevent redundant selections in the final keyword set, Carbonell and Goldstein [6] introduce Maximal Marginal Relevance (MMR), which balances the relevance to the chunk with the diversity among selected phrases. Without MMR, the similarity ranking would produce redundant output.

Yepes et al. [41] apply this principle directly in a RAG pipeline for financial reports: after element-based chunking, up to six representative keywords per chunk are generated via GPT-4 and stored as metadata within the chunk ("Keywords Chipper"). Their evaluation, however, does not isolate what the keywords contribute. The keyword variant is compared against fixed-size baselines rather than against the same element-based chunks without keywords, and it is the weakest of the three metadata types they test: on page-level retrieval accuracy it reaches 46.1 % against 62.4 % for chunk summaries and 67.4 % for the prefix-and-table-description variant, and stays below every fixed-size baseline (68.1–73.1 %). On end-to-end question answering the ordering reverses and the keyword-enriched

chunks surpass the best fixed-size baseline (53.2 % against 48.2 % manually rated accuracy). Whether keyword injection improves *retrieval* therefore remains open in their design, which is the comparison our work provides.

This thesis implements metadata injection at the chunking stage: V3a augments the chunks with KeyBERT keywords, using exactly the POS-filtered candidate generation and MMR diversification described above (configuration detailed in Section 4.3) while V3b uses LLM-generated keywords. These additions affect both the dense embedding and the BM25 index.

3.3 RAG-Based AI Tutoring Systems

RAG performance varies substantially across domains and retrieval quality is decisive for answer quality: Magesh et al. [28] find that commercial RAG-based legal research tools hallucinate between 17% and 33% of the time, and characterise failures along *correctness* and *groundedness*, showing that hallucination is also triggered by retrieval failures and not the generator alone. Oche et al. [30] list retrieval quality among the persistent challenges of deployed RAG systems across domains.

Within education, recent systems invest almost exclusively on the generation side. Jain et al. [16] present KITE, a RAG-based tutoring engine for algorithmic reasoning whose intent-aware Socratic strategy classifies student queries and answers with guiding questions instead of solutions. Li et al. [26] combine Knowledge Tracing with RAG in TutorLLM, tailoring each answer to a predicted learning state and reporting gains in user satisfaction and quiz scores over a general-purpose LLM. Reza et al. [34] address cost, privacy and control concerns of cloud deployments with small, locally hosted LLMs, a teacher-in-the-loop refinement loop and a verifier LLM, demonstrated in a college physics course on institutional hardware.

These systems share a common dependency: the quality of their responses is bounded by the quality of the retrieval layer, but the investment is made on the generation side (Socratic prompting, knowledge tracing, verifier loops) while taking retrieval quality mostly as given. This thesis addresses that dependency directly, isolating and improving the retrieval component for a C programming course corpus.

4 Methodology

The methodological implementation of this work follows an incremental evaluation model. The three iterations (V1, V2, and V3 divided into 3a and 3b) are different ingest versions that each improve one aspect of data preparation. We evaluate the impact every stage of preparation has on retrieval quality in the context of C programming. This work applies the Data-Centric AI paradigm described in Section 2.4: the model and retrieval architecture are held constant across all experiments, while the data preparation pipeline is iterated. By iterating we can measure the effect on the quality of each change independently. V1 is a naive baseline that represents the current unoptimised state of the Arcana deployment. V2 addresses the primary type of failure of naive chunking: context fragmentation, where semantically coherent passages are split across chunks. V3 addresses the vocabulary gap between short, colloquial student queries and longer instructional text, which structural chunking alone cannot close. This gradual data refinement follows the iterative data engineering loop proposed by Zha et al. [42]: inspect the data, identify the dominant error type, refine the representation, and re-evaluate on a fixed benchmark.

Our methodology is organised by the following sections: Section 4.1 describes the fixed five-phase evaluation pipeline shared by all versions. Section 4.2 introduces the three corpus source types. Section 4.3 defines the three data-preparation stages (V1–V3) that are the independent variable to be evaluated and Section 4.4 describes the comparative analysis between with and without the CTC. Section 4.5 explains the construction of our gold-standard benchmark, and Section 4.6 defines the chunk-level metrics we introduce to exploit our specific benchmark.

4.1 Architecture

The evaluation system implements a five-phase pipeline shown in Figure 1. All versions share the same fixed infrastructure: the embedding model is `Qwen3-Embedding-4B` [43] (dimension 2560), accessed via the OpenAI-compatible *academiccloud* API of the GWDG. All versions use hybrid retrieval combining dense (Qwen3) and sparse (BM25 via `rank_bm25`) components, fused via RRF [9] with $k = 60$.

The RAG system whose ingest this work optimises is *Arcana* (Section 1), whose current configuration could not host the experiments of this thesis for two reasons. First, the ingest pipeline provides no possibility for adding section paths or a keyword header to a chunk, so two of the three preparation stages evaluated here could not be surveyed at all. Second, its index groups chunks in vector space and restricts the search to a single group, so that relevant chunks assigned to another group cannot be retrieved regardless of how the data is prepared.

The evaluation therefore runs on *zVec* [1], introduced in Section 1. It is an in-process vector database written in C++ and published by Alibaba Group under the Apache-2.0 licence, offering flat, Hierarchical Navigable Small World (HNSW), and quantised HNSW indexes. Decisive for this study is that its search covers the index as a whole, so every chunk competes for a rank in every query and a measured difference between two versions can be attributed to the data preparation. zVec is not part of the current Arcana deployment, but the evaluated configurations serve as an implementation example for a future optimised Arcana ingest.

Figure 1: Five-phase evaluation pipeline. The three data preparation stages (V1–V3b) are only distinct in Phase 1; all downstream phases share a fixed retrieval and evaluation infrastructure. Orange dashed arrows indicate the caching to disk: repeated runs skip these steps and load from cache. The corpus-hash (`.sha256`) ensures that a change in chunk texts triggers a full embedding rebuild.

4.2 Data Sources

The corpus used for the study consists of three different source types from university teaching materials:

- **Lecture script (PDF):** Hierarchically structured documents (decomposed from one 206-page script document) with code examples and mathematical definitions. Converted to Markdown via the SAIA Docling API, filenames encode the corresponding script segment and page range (e.g. `programmierung_mit_ansi_c-29-40.pdf`).
- **SmartBeans exercises (YAML):** 96 structured files, each containing a programming task with fields `title`, `shortname`, `task`, `solution`, and `courses.ckurs.tags`.
- **Curated Tutoring Collection (YAML):** A manually curated collection of 19 articles (`topic`/`content`) covering environment setup (WSL, VS Code, GCC), the SmartBeans platform, exam information, and C-specific topics (memory management, pointers, strings, I/O, common error messages). The Curated Tutoring Collection *fills informational gaps* in the corpus and is currently not part of the deployed system: it contains knowledge that students query which is genuinely absent from the course documents. From a DCAI perspective, curating this collection is a data preparation decision that fills gaps in the raw corpus rather than compensating for them at the model level. The isolated contribution of this corpus extension to overall retrieval quality is measured via the comparative analysis described in Section 4.4.

The design of our CTC allows the addition of missing data, which makes data preparation an iterative process in deployment. It can be extended dynamically as student questions come up that the existing course materials cannot answer. Each article in the CTC addresses information gaps identified during the corpus analysis of student queries from tutorials. This extension is the DCAI feedback loop that the comparative analysis in Section 4.4 intends to make measurable.

4.3 Evolution Stages of the Data Ingest

Between versions, we evaluate the following stages of data preparation:

4.3.1 V1: Naive Baseline

The first stage serves as a reference point and applies the simplest reasonable extraction strategy to each source type, without taking advantage of its structure:

- **Lecture script (PDF)** Conversion via *Docling* [5] to Markdown (via SAIA Docling of the GWDG), followed by *fixed-size chunking* with a window of 500 words without overlap.
- **SmartBeans and CTC (YAML):** All YAML files are read as raw strings without special parsing. The raw structural keys (`title:`, `task:`, `solution:`, `question:`, `answer:`, ...) remain as noise in a single block per file. V1 is naive across all source types, therefore the CTC also appears as a single unstructured block, where information is mixed.

The result of naive chunking is the segmentation of related information into different blocks and the loss of the field structure of the YAML files. The resulting *context fragmentation* problem [38] serves as the baseline against which the following versions are measured. Following [23] and [37], 500 words is adopted as the standard fixed-size default. Kochanke finds that fixed-size chunking achieves competitive precision but underperforms on recall and nDCG because semantically coherent passages are separated. V2 addresses this by replacing the fixed window with heading-aware structural chunking.

4.3.2 V2: Structural Chunking with Hierarchy Embedding

The second stage addresses the maintenance of structure in the source documents. The parser operates based on the file type:

- **Lecture script (PDF):** The script is converted to Markdown and split using a heading-aware parser. Code blocks are always kept intact and never divided across chunk boundaries. We targeted a chunk size of 500–1500 words to balance retrieval precision and contextual coverage [23]. However, because the actual chunk lengths are adjusted to document structure rather than the theoretical chunk size, a section boundary (e.g., a new heading or a YAML exercise block) immediately forces the creation of a new chunk (`force=True`). The minimum threshold (`MIN_TOKENS=500`; the parameter name is historical, the unit is words) is only used *within* a continuous section. In practice, this leads to a median chunk size of 215 words across the whole V2 corpus, with the majority of chunks falling below the 500-word target. That corpus-wide median is, however, a mixture of the two source types and understates the size of the chunks described in this bullet: the PDF chunks reach a median of 380 words and therefore stay reasonably close to the intended target, whereas the task/solution split of the YAML exercises produces a median of only 96 words (Table 1). This design choice was maintained to prevent unrelated information from being artificially put into the same chunk. Overlap between chunks is set to 20 % of the word count of the neighbouring chunk [23, 16]. Overlap prevents related information from being fragmented, which would otherwise lower the quality of the resulting vector embeddings [3]. During parsing, a section stack records the hierarchical position of each chunk in the document (e.g. `Chapter 12 > 12.4 Pointers and const`). This section path is written as a Markdown header in front of the chunk text before embedding, so that it is received as token material. Adding structural context is treated here as part of structural chunking, not as a separate enrichment step [5]. The full chunker is reproduced in Listing 1 (Appendix A).

- **SmartBeans (YAML):** Each SmartBeans task is separated into two chunks: a task chunk and a solution chunk (only if the solution exists and is not marked as TODO). All header labels are kept in German, because the corpus and student queries are in German. The task chunk receives a structured header with the fields `Titel:` (title), `Thema (Kurz):` (shortname), `Themen:` (tags), and `Aufgabenstellung:` (problem statement); the solution chunk is prefixed with `Musterlösung (C-Code):`. This labelling provides context and German labels ensure vocabulary alignment with the query language.

- **CTC (YAML):** Each of the 19 articles is mapped to a single chunk with the format `{topic}\n{content}`.

What V2 changes, and what that permits us to attribute. V2 is not a single manipulation but a bundle of preparation decisions applied together: heading-aware chunk boundaries instead of a fixed window, atomic code blocks, a 20 % overlap between neighbouring chunks, the section path prepended as a Markdown header, the field-structured task/solution split of the YAML exercises with German field labels, and, as a consequence of all of these, a smaller median chunk size. The experiment is designed to measure the effect of this bundle against the naive baseline, and the granularity control (Section 4.3.3) only separates chunk size from the rest. It does not separate the remaining preparation decisions from each other. Whenever the results and the discussion attribute an effect to "structural chunking", what is attributed is this group of decisions without the size effect, not the heading boundaries in isolation. Section 6.5 states the consequence of this methodological decision.

Markdown-aware chunking with section boundaries consistently outperforms token-based chunking on recall and nDCG and prepending the hierarchical section path to each chunk leads to retrieval improvements [23]. However, Kochanke also finds that breadcrumb path metadata (equivalent to Hierarchy Embedding in this case) alone is typically insufficient to close the full vocabulary gap between colloquial student queries and instructional text, a limitation that motivates the keyword injection in V3.

4.3.3 V1-215: Granularity Control Baseline

Moving from V1 to V2 changes two things at once: chunks become structure-aware *and* considerably smaller (median 215 instead of a fixed 500 words). To determine whether metric changes between V1 and V2 are an inevitable consequence of smaller chunk sizes or a real effect of structural chunking, an additional granularity control baseline (V1-215) is added. Unlike V1's 255 chunks of 500 words, this baseline applies V1's structure-blind fixed-size splitting at a smaller 215-word window, which is the median chunk length of V2 and matches V2 almost exactly in granularity (454 vs. 451 chunks; median 215 words each). Because V1-215 uses V1's naive YAML chunking, this match holds for the index as a whole but not for its composition: the 215-word window applies only to the lecture script, which V1-215 therefore splits more finely than V2 (357 vs. 241 chunks), while each exercise file remains a single block and the exercises stay coarser than in V2 (97 vs. 210 chunks). Table 1 reports the resulting distributions and Section 6.5 discusses what this means for the control. V1-215 is not a pipeline stage of its own: it shares V1's naive extraction, runs through the same fixed retrieval and evaluation pipeline (Section 4.1) and is used wherever the effect of chunk size has to be separated from the effect of structure: the V1 $\rightarrow$ V2 comparison and, in RQ4, the CTC queries.

Table 1: Chunk statistics per ingest version, measured on the cached chunk files of the full corpus (including the CTC). Word counts are whitespace-separated tokens, the unit the chunker operates on. The corpus-wide median of 215 words that motivates the granularity control (Section 4.3.3) is a mixture of two very different source types: V2's PDF chunks are considerably larger and its YAML chunks considerably smaller than that single figure suggests. Excluding the CTC, which is the primary evaluation condition (Section 4.4), the indexes contain 254 (V1), 453 (V1-215) and 432 chunks (V2, V3a, V3b).

	All chunks		PDF (lecture script)		YAML (exercises, CTC)	
Version	n	Median	n	Median	n	Median
V1 (500 w)	255	500	158	500	97	485
V1-215 (control)	454	215	357	215	97	485
V2 (structural)	451	215	241	380	210	96
V3a (KeyBERT)	451	226	241	391	210	107
V3b (LLM)	451	227	241	394	210	109

4.3.4 V3: Keyword Injection

V3 is identical to V2 in terms of chunking and retrieval, but it adds a semantic enrichment of the chunks: for each chunk, a small set of domain-specific keywords is extracted (a fixed 7 in V3a, between 5 and 10 in V3b) and inserted as an additional header (`Keywords: ...`) between the section path and the chunk body; no chunk is split, merged, or dropped in this step, so V2, V3a, and V3b share the same 451 chunks position for position (241 PDF, 210 YAML), and the header is the only difference between them. This also explains the small rise in median chunk length from V2 to V3a/V3b in Table 1: it is entirely the injected header text (the keyword line itself plus a small number of words for the label), not a change in chunking. The goal of the enrichment is to close the *semantic gap*[42, 37] between short, imprecise student queries and longer text chunks from the source texts: for example, a question like "When do I use & and when *?" does not contain the technical terms (address operator, dereference, pointer) that appear in the relevant teaching passage. Explicit keyword headers address the gap between query and document vocabulary [23].

Two extraction strategies are evaluated:

V3a — KeyBERT with POS-based candidate extraction. Version 3a uses KeyBERT [12] with the POS-based candidate filter introduced by Schopf et al. [36] (Section 3.2.3): only grammatical noun phrases are admitted as candidates, which is intended to prevent syntactic and code tokens from entering the candidate set; Section 6.3 shows that this filter does not hold on the corpus used here. Because of our German corpus, the German spaCy model `de_core_news_sm` [14] is used for POS annotation. From the filtered candidates, KeyBERT selects 7 keywords per chunk via MMR [6] with $\lambda = 0.5$, balancing relevance and diversity. The internal embedding model is `paraphrase-multilingual-MiniLM-L12-v2` [33], a local CPU model chosen for cost and reproducibility.

V3b — LLM-generated. Keywords are generated by a Large Language Model (`qwen3-30b-a3b-instruct-2507` via the academiccloud API) using a German prompt (see Appendix E), which asks the model to return a JSON list of 5 to 10 domain-specific C

terms per chunk. Compared to Version 3a, this approach promises higher precision since the LLM understands technical language in context. It can find synonyms and concepts not literally described in the chunk text. The trade-off is higher cost, but only once at index-time, dependence on model and prompt choice and the risk of hallucinated terms. The focus of the comparison of V3a and V3b is evaluating the effectiveness of switching from a lightweight, statistical keyword strategy to a more expensive, LLM-based strategy.

4.4 Comparative Analysis: With and Without CTC

Including the Curated Tutoring Collection in the corpus represents two different effects: the different performance on standard course material and the additional coverage provided by the collection. To isolate these effects, each pipeline version is run twice: once without the CTC as the primary comparison condition and once with it. This delivers the eight main configurations in total (the granularity-control baseline V1-215 described in Section 4.3.3 is additionally run in both conditions):

`v1_baseline`	`v1_baseline_no_CTC`
`v2_chunking`	`v2_chunking_no_CTC`
`v3a_keywords`	`v3a_keywords_no_CTC`
`v3b_llm_keywords`	`v3b_llm_keywords_no_CTC`
`v1_215`	`v1_215_no_CTC`

When the CTC is excluded, chunks from `curated_tutoring_collection.yaml` are filtered from the corpus (the chunk cache always contains the full corpus for cache compatibility). Queries whose references point exclusively to the CTC ($n = 11$) are reclassified as `out_of_scope` internally.

The primary comparison is the **without-CTC** configuration (V1 → V2 → V3a / V3b), which allows us to refer the measured differences only to data preparation. The **with-CTC** configuration is reported additionally as an insight of realistic system performance, since the collection represents a useful part for the future deployed corpus. The difference between both configurations quantifies how the retrieval quality can be improved by closing informational gaps through supplementary material. A secondary effect is also visible in the V1 → V2 comparison: in V1 the entire collection is not structured at all, while V2 splits it into 19 individually retrievable articles. The comparative analysis distinguishes whether the V2 gain on CTC queries is because of the improved CTC structure or from better chunking of the lecture material.

4.5 Gold Standard Benchmark Construction

Since no public evaluation dataset exists for the C programming course at the University of Göttingen, a dedicated benchmark was constructed within the scope of this work. Its design follows the approach of other domain-specific RAG benchmarks as those for financial documents [37, 41], transferred to the educational context of the C programming course.

4.5.1 Source of Queries

The benchmark queries were collected over a period of two weeks during the C programming intensive course in the winter semester break in March 2026. The course tutors recorded the questions that were asked by students during their tutorial sessions. Bullet points were rephrased into full-sentence questions and full queries were preserved in their original

form, including colloquial formulations and unspecific references to tasks ("why does my code not work?"). This linguistic authenticity is methodologically important, because it is exactly the gap between colloquial and technical language in the lecture script that is the core challenge which a teaching tutor system must cope with.

The collected questions cover different thematic fields of the material: from toolchain setup (WSL, VS Code) through language fundamentals (loops, functions) to advanced topics (pointers, memory management, structs).

4.5.2 Annotation Process

The annotation process was carried out in two stages:

1. **Annotation by the author:** For each collected query, the relevant source documents (lecture script passage, SmartBeans tasks, CTC articles) were identified, the matching evidence text was extracted as a verbatim substring and a matching score was assigned according to the scale defined below.
2. **Validation by the course instructor:** After the annotation by the author, the course instructor verified the accuracy of the references and the matching scores, and approved the final annotation set.

4.5.3 Annotation Structure

The following annotations were made per query:

- `type`**:** There are two categories of retrieval requirement: *Redundant* and *Complementary*, both defined below.
- `references`**:** A list of up to 14 gold references, each consisting of:
  - `gold_id` — filename of the relevant source document,
  - `evidence` — the relevant text passage as a substring of the original document,
  - `matching_score` $\in [0.0, 1.0]$ — relevance score (see below).

Score scale. The score scale was preferred over a binary relevance rating because in the educational context the majority of queries have multiple relevant sources that are, however, differently *essential* for the answer. The scale therefore has the following meaning:

- 1.0 — The reference contains the core information, without it the question cannot be answered.
- $[0.8; 1.0)$ — The reference contains important information, it is a "must-have" for a complete answer.
- $[0.5; 0.8)$ — The reference is helpful but redundant to higher-scored references or secondary for answering the question.
- $(0.0; 0.5)$ — The reference is helpful but not needed for an adequate answer.

In this work, the threshold ≥ 0.8 is used as the boundary for *highly relevant* references (G_{high}) (see Section 4.6.1).

Query types. Each query is labelled with a type that identifies the retrieval requirements:

- **Redundant (29 queries):** The answer is fully contained in a single document; further references cover the same content redundantly. These queries test whether the system finds at least one adequate source. Of the 29, eleven are answered only by the CTC; without the CTC these queries become out-of-scope, reducing the redundant subset to $n = 18$ in the without-CTC condition.
- **Complementary (54 queries):** A complete answer requires the retrieval of multiple complementary documents and passages. These queries test the multi-hop ability of the retriever.
- **Out-of-scope:** Queries whose answer is not contained in the corpus. With the full corpus (`INCLUDE_CTC = True`), the benchmark contains no explicit OOS queries. With the collection excluded (`INCLUDE_CTC = False`), 11 CTC-only queries are then reclassified as OOS at evaluation time.

In total, the benchmark processes **83 queries with 342 gold references** pointing to 57 distinct source documents: 18 PDF script segments, 38 SmartBeans YAML exercises, and the CTC, which carries 27 of the references. The mean matching score across all references is 0.82 (min: 0.30, max: 1.00).

4.5.4 Evidence-Coverage Validation

Before evaluation, we check whether each `evidence` text used in the benchmark can be located in at least one chunk of the expected document. This validation step uncovers errors in document conversion as well as chunk boundaries that split evidence. References whose evidence cannot be found in any chunk of the expected document are marked as `MISSING`. All references with a non-empty evidence string are included in the evaluation, those *without* are skipped.

In V1 (500-word windows), 4 evidence snippets are structurally marked as `MISSING`, because the window boundary cuts through the relevant passage; the queries carrying them are marked `evidence_infeasible`. In the granularity control V1-215 (Section 4.3.3) the same mechanism affects 17 queries, because a 215-word window cuts more often than a 500-word one. V2 and V3 do not have this problem by construction, since their chunking is structure-aware.

How infeasible queries are treated. Two decisions follow from this and are applied consistently throughout Section 5. *Levels* (the per-version metric values in Table 3 and in all bar charts) are computed over every in-scope query of the condition ($n = 72$ without the CTC), with an infeasible query contributing the score it actually achieves, which is zero for the references that cannot be located anywhere in the index. Structural infeasibility is a property of the data preparation under evaluation, so a version that cuts evidence apart is charged for it rather than having the affected queries removed from its own score. *Paired differences* (the forest plots and every reported Δ with a bootstrap CI) are computed on the subset of queries that are structurally solvable in *both* compared versions, because a paired test over a query that one side cannot solve by construction would count the same preparation defect a second time. The V1 $\rightarrow$ V2 comparison therefore runs on $n = 68$ while the corresponding levels are reported over $n = 72$; the applicable n is stated in the caption.

One consequence is that V1-215, which is affected four times as often as V1, enters the level comparison of Figure 4 with a handicap that is not a ranking effect. Section 6.5 states what that means for the control.

Data availability. The benchmark (`benchmark.json`: 83 queries, 342 gold references), the Curated Tutoring Collection (19 articles, authored for this thesis), and the full pipeline code (ingest, retrieval, evaluation) are published at `https://gitlab.gwdg.de/corinna.liley/c-data-centric-ai-thesis` to allow independent verification of the results reported in Section 5 and reuse of the benchmark for future work on this or comparable courses. The lecture script and the SmartBeans exercises are course material of the University of Göttingen and are not included.

4.6 Evaluation Metrics

The retrieval quality of each pipeline configuration is measured by two complementary metric approaches: standard ranking metrics applied at the document level (defined in Section 2.2) and score-weighted chunk-level metrics that exploit the graded annotation of our benchmark. Although nDCG is typically applied at the document level in IR evaluation (Section 2.5), it is applied at chunk level throughout this work (Section 4.6.1). When mentioning nDCG in the following, we refer to this chunk-level variant. The two metric families are not directly comparable because they operate at different granularity levels: chunk-level metrics operate over the top-k retrieved chunks, whereas the document-level metrics operate over the top-k documents. We evaluate on both levels, but focus on chunk-level because document-level metrics can produce false positives: a system that returns a chunk from the correct source document but not the specific passage containing the answer is counted as a success at the document level while failing to find the actual information. Figure 2 shows a hard-negative scenario from our evaluation corpus.

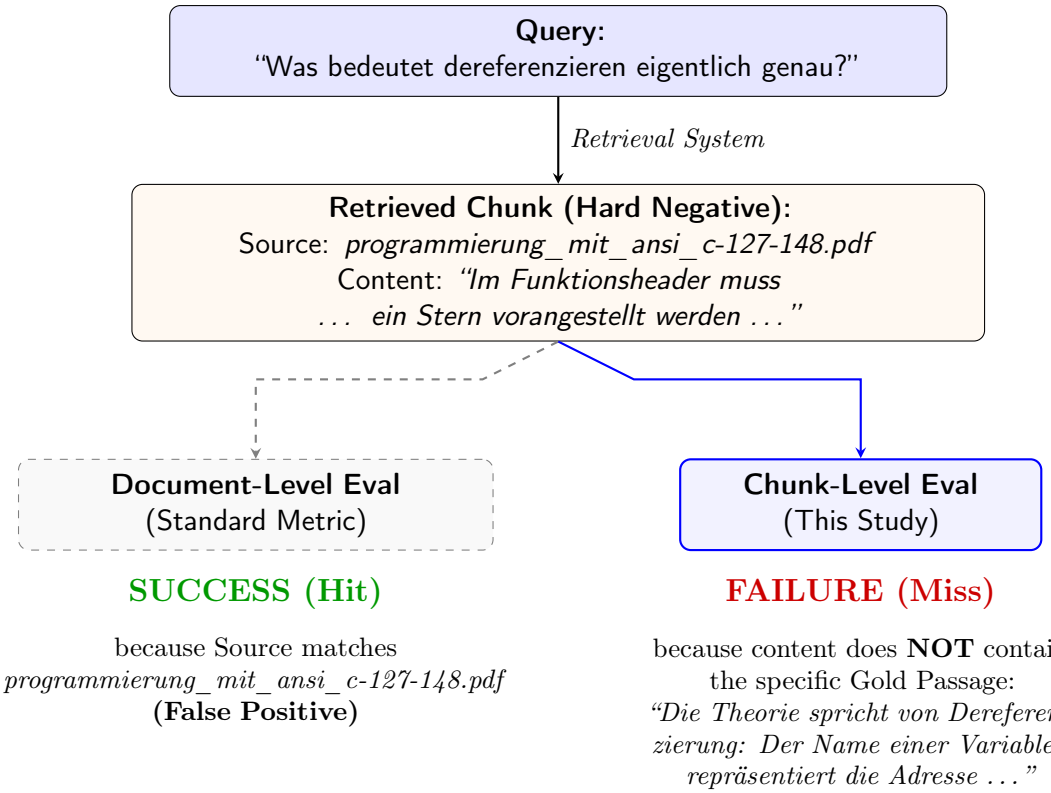

Figure 2: Hard-negative scenario from the evaluation corpus. The retrieval system returns a chunk about pointer syntax from the correct source document (*programmierung_mit_ansi_c-127-148.pdf*, 23 chunks), but not the chunk containing the theoretical definition of dereferencing. A document-level metric counts this as a success (false positive), chunk-level evaluation correctly identifies it as a miss.

4.6.1 Score-Weighted Chunk-Level Metrics

Since the benchmark assigns each evidence a `matching_score` $\in [0.0, 1.0]$, classical recall and MRR metrics can be extended by score weighting. Let $G_{\text{high}} = \{r \in R : \text{matching_score}(r) \geq 0.8\}$ be the set of highly relevant ("must-have") references for a query and let rank_i be the rank of the first chunk containing the evidence of reference i in the top-k list, counted from 1 (∞ if not found). The following set therefore combines adopted and newly defined measures. nDCG@k, Recall@K and MRR are established measures [29, 19, 40], applied here to chunks instead of documents. Weighted Relevance Score (WRS@k), Weighted Recall (WRec@K), Chain-MRR and Strict High-Score Hit@K are defined for this thesis: they exploit the graded annotations while implementing the same reciprocal-rank and recall principles, and are designed to isolate *whether* evidence is retrieved from *how early* it is retrieved. Their behaviour is validated empirically at the end of this section and their granularity dependence is examined in Section 6.1.

- **nDCG@k**: *How well does the ranking order reflect relevance grades?*

  Unlike the document-level variant (Section 2.2), ranking operates over chunks with graded relevance and IDCG includes all gold references regardless of retrieval outcome:

$$\text{rel}(c) = \max_{e \in E_c} \text{matching_score}(e) \tag{6}$$

where E_c is the set of gold references whose evidence was found in chunk c; co-located references take the maximum score, not the sum. IDCG [40] is built from all gold reference scores in descending order, regardless of whether they were retrieved. A missing reference therefore appears in the denominator but not the numerator, preventing artificially inflated nDCG values when corpus coverage is poor.

Keeping every reference in the denominator while rel(c) takes their maximum is what makes unretrieved evidence visible, but it also ties the achievable maximum to the chunking: a query whose references share one chunk cannot reach nDCG = 1.0, and that maximum rises as chunks get finer. nDCG levels are therefore read within a granularity here, Section 6.5 states what this means for the V1 → V2 step.

- **Weighted Relevance Score (WRS@k):** *How well does the system find and rank all relevant evidence?* (implementation in Listing 2, Appendix A)

$$\mathrm{WRS@K}(q) = \frac{\sum_{i \in \mathrm{refs}(q)} s_i \cdot \mathrm{decay}(\mathrm{rank}_i)}{\sum_{i \in \mathrm{refs}(q)} s_i} \tag{7}$$

with s_i = matching_score(r_i) and decay(rank) = 1/rank (reciprocal decay). A high-scoring evidence at rank 1 contributes its full weight; at rank 10 only one tenth. References without an evidence snippet are skipped; references not found contribute zero to the numerator.

- **WRec@K:** *What proportion of the weighted relevance mass is findable at all in the top-k?*

$$\mathrm{WRec@K}(q) = \frac{\sum_{i \in \mathrm{refs}(q)} s_i \cdot \mathbf{1}[\mathrm{Evidence}_i \in \text{Top-K}]}{\sum_{i \in \mathrm{refs}(q)} s_i} \tag{8}$$

Unlike WRS, there is no rank decay. As with WRS, only references with an explicit evidence snippet are included; references without a snippet are skipped.

- **Recall@K ($G_{\mathbf{high}}$ only):** *Are all must-have references retrievable within the top-k?*

$$\mathrm{Recall@K}(q) = \frac{|\{i \in G_{\mathrm{high}}(q) : \mathrm{Evidence}_i \in \text{Top-K}\}|}{|G_{\mathrm{high}}(q)|} \tag{9}$$

Applicable only to queries with at least one G_{high} reference.

- **Chunk-MRR:** *How quickly does the system find anything useful?* The reciprocal rank of the first relevant chunk found across all references.

- **Chain-MRR (complementary queries only):** *How deep must the system search before the last required fragment is found?* (implementation in Listing 3, Appendix A)

$$\mathrm{ChainMRR}(q) = \begin{cases} 1/\max_{i \in G_{\mathrm{high}}(q)} \mathrm{rank}_i & \text{if all } G_{\mathrm{high}} \text{ refs found} \\ 0.0 & \text{if at least one } G_{\mathrm{high}} \text{ ref missing} \end{cases} \tag{10}$$

This bottleneck variant addresses complementary queries whose complete answer requires several independent chunks simultaneously. Cases where the last required fragment appears far down the ranking will have a low score. Together with Chunk-MRR (best case: first hit), Chain-MRR (worst case: last necessary hit) covers the full retrieval depth.

- **Strict High-Score Hit@K:** *Does the system retrieve every must-have reference within the top-k?*

  A binary per-query criterion: 1 only if all G_{high} references are found, otherwise 0. Aggregated as a count over applicable queries (e.g. 41/66 for V1). Applicable only to queries with at least one G_{high} reference.

Queries without any G_{high} reference return `None` for metrics that operate on G_{high} and are excluded from the corresponding mean, the applicable count n_{appl} is reported separately.

Empirical validation of score-weighting. The WRS design assumes that late retrieval of high-relevance evidence should be penalised more heavily than late retrieval of secondary evidence. This assumption is supported by the data: high-value references ($s_i \geq 0.8$) are retrieved at better ranks than lower-weighted ones (median rank 3 in V1 and 4 in V2, versus 6 and 5 for lower-weighted references; see Figure 12 in Appendix B). The evidence that matters most tends to surface first, indicating that the 1/rank decay is a reasonable penalty for this corpus.

4.6.2 Out-of-Scope Handling (AUROC)

A practical tutor system must be able to reject queries when the answer cannot be retrieved from the corpus. Magesh et al. [28] show that RAG-based systems produce hallucinated or unsupported answers especially when retrieval fails to find the relevant information, but they also count refusals and unresponsive answers as failures, so rejection is a trade-off rather than a fix. Since semantically related queries and chunks lead to nearby vectors [3, 23], we hypothesise that a retrievable answer leads to a high cosine similarity score between the query and at least one corpus chunk, while an OOS query for which no relevant chunk exists produces a low score. If so, we can use a score threshold in a deployed tutoring system to decide whether to answer or decline.

To measure how well the two cases can be separated, OOS queries are passed through the full retrieval pipeline and only their `best_score` is recorded, that is the cosine similarity between query and the chunk that the RRF fusion places at rank 1. Since the fused ranking can promote a chunk on the strength of its BM25 score, this is not necessarily the maximum cosine similarity in the index; it is the score of the chunk the deployed system would actually act on, which is the quantity a refusal rule would have to threshold. The **AUROC** (Area under the ROC Curve [10]) between in-scope and OOS best scores is statistically equivalent to the Wilcoxon test [13]:

$$\text{AUROC} = \frac{|\{(p,n) : p > n\}| + 0.5 \cdot |\{(p,n) : p = n\}|}{|P| \cdot |N|} \tag{11}$$

with P = in-scope best scores, N = OOS best scores. It counts for every pair (p, n) how often the in-scope score exceeds the OOS score. AUROC = 1.0 means perfect separability: every in-scope score is higher than every OOS score and a threshold exists that correctly classifies all queries.

It must be noted that the OOS sample in this benchmark is small: with the Curated Tutoring Collection excluded, only 11 queries are reclassified as out-of-scope. The results should therefore be interpreted as a first insight rather than definitive evidence. Whether score-based thresholding is a reliable rejection mechanism in practice would require a much larger and more representative OOS query set and remains a direction for future work.

5 Results

The results show a central paradox. Data preparation enhances the document-level Hit@1 and chunk-level nDCG, but the weighted metrics, especially WRS (*how early* relevant evidence is read), do not rise consistently. RQ2 and RQ3 explain why: they show where the improvement actually lands and what the document-level gain means. RQ4–RQ6 examine what impact corpus extension, query types, and document types have across versions. RQ7–RQ8 address deployment: out-of-scope calibration and latency.

As defined in Section 4.1, all four pipeline versions share the same retrieval pipeline: `Qwen3-Embedding-4B` embeddings, hybrid dense + BM25 retrieval fused via RRF ($k = 60$), and a top-$k = 20$ cut-off (at chunk-level). We choose this value because queries carry up to 14 gold references (median 4), so k=20 provides sufficient allowance to retrieve all required evidence pieces. Only the ingest stage varies. Unless stated otherwise, results refer to the *without-CTC* condition ($n = 72$ in-scope queries), which holds the corpus constant across versions. The Curated Tutoring Collection is reintroduced only in RQ4.

Evaluation is mainly at chunk-level, where gold references point to specific passages, not only documents. Hit@1 is reported merely as a baseline. Since a query requires a median of four references across two different documents, retrieving just a single document is useful as a starting point, but not enough for a complete answer, and it is hard to judge from a document-level hit alone whether a generator could produce a grounded answer from that whole document. Hit@5 is likewise computed and reported only in Appendix F (Table 7) for completeness, and it is referred to in Section 6.4 where it qualifies the recommendation. Hit@10 and Hit@20 are computed by the pipeline but not reported anywhere in this thesis since they show no additional value across all versions. All reported effects are accompanied by 95 % paired bootstrap CIs (2000 resamples, seed 42); an effect is called *significant* only when the interval excludes zero. These paired intervals are produced by the analysis stage; the per-run intervals that the evaluation pipeline itself computes (Phase 5, Figure 1) use 1000 resamples at the same seed and are not reported here.

For a better understanding of the following figures, Table 2 maps each metric to the question it answers (reflects Section 2.5).

Table 2: Evaluation metrics and the question they answer. All chunk-level @k metrics are evaluated at k=20. Chunk-MRR and Chain-MRR are rank-based but computed within the same top-20 list.

Metric	Level	Answers the question…
Hit@1	doc	Is the top-ranked document relevant at all? *(necessary, not sufficient)*
nDCG@20	chunk	How good is the overall ranking of the graded relevant chunks?
WRS@20	chunk	How *early* are the score-weighted evidence chunks retrieved?
WRec@20	chunk	What fraction of the score-weighted evidence is retrieved at all?
Recall@20	chunk	What fraction of the must-have (G_{high}) references is retrieved at all?
Strict@20	chunk	In what share of queries are *all* must-have references retrieved?
Chunk-MRR	chunk	At what rank does the *first* relevant chunk appear?
Chain-MRR	chunk	How deep must we search for the *last* required (G_{high}) piece?
AUROC	—	Can the system separate answerable from out-of-scope queries?
Latency	—	What does a query cost (embedding / local / end-to-end)?

5.1 Impact of Iterative Preparation

RQ1. *Does an iterative data-preparation approach consistently improve retrieval quality?*

Primary metrics (Table 2): Hit@1, nDCG, WRS, Chain-MRR.

Figure 3 and Table 3 give the answer, and it is not uniform across metrics. From V1 to V3b, document-level Hit@1 rises from 0.569 to 0.681 and chunk-ranking quality (nDCG) from 0.412 to 0.546, both trending upward, and achieved without touching the model or the retriever. However, WRS does not improve ($0.303 \rightarrow 0.300$) and Chain-MRR drops before recovering ($0.204 \rightarrow 0.182 \rightarrow 0.164 \rightarrow 0.218$). This is the paradox this chapter examines: measuring *whether* one right document is ranked at the top and *how well* chunks are ranked, data preparation succeeds, whereas measuring *how early the weighted evidence is actually read* (WRS), it does not. RQ2 and RQ3 explain why.

The two coverage metrics are the reason no version is best on everything: Recall@20 and Strict@20, which ask only whether the must-have references reach the top-20 at all, both peak at *V2* (0.816 and 0.667), where V3b reaches 0.812 and the same 0.667. The iterative approach improves retrieval quality overall, but for different metrics at different stages rather than monotonically on every measure.

Table 3: Core metric overview (without CTC). The columns do not share one sample: Hit@1, nDCG, WRS and WRec are averaged over all $n = 72$ in-scope queries, Recall@20 and Strict@20 over the $n_{\text{appl}} = 66$ of them that carry at least one G_{high} reference, and Chain-MRR over the $n_{\text{appl}} = 50$ complementary queries for which it is defined. All values are ratios in $[0, 1]$; Strict@20 is the share of queries in which every G_{high} reference is retrieved. Best value per column in bold. The full table including MRR and Hit@5, and the V1-215 control, is given in Table 7 in the appendix.

	Doc-level	Chunk-level					
Version	Hit@1	nDCG	WRS	WRec	Recall@20	Strict@20	Chain-MRR
V1 (500 w)	0.569	0.412	**0.303**	0.706	0.787	0.621	0.204
V2 (structural)	0.597	0.526	0.287	**0.752**	**0.816**	**0.667**	0.182
V3a (KeyBERT)	0.597	0.521	0.274	0.733	0.798	0.636	0.164
V3b (LLM)	**0.681**	**0.546**	0.300	0.749	0.812	**0.667**	**0.218**

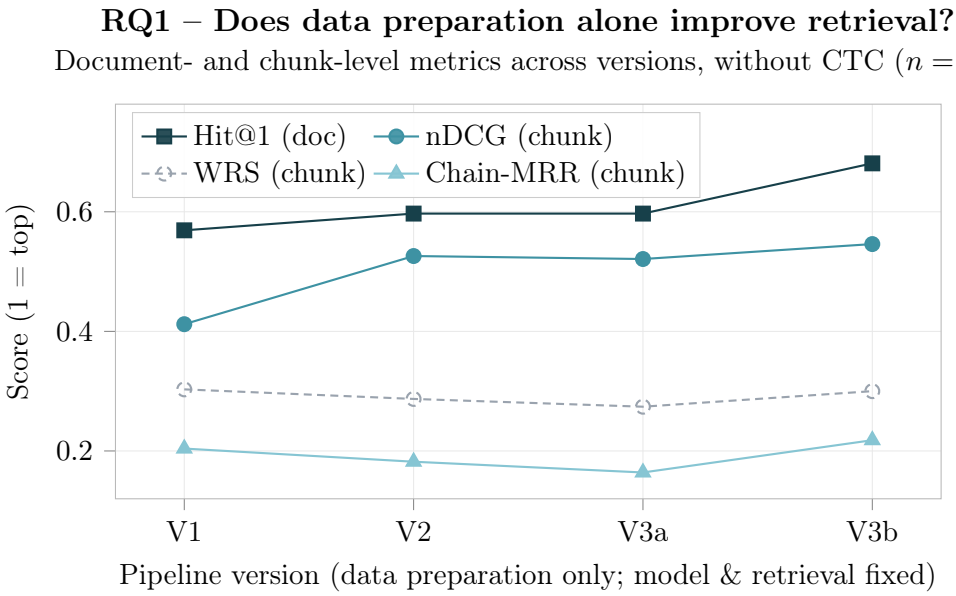

Figure 3: Document- and chunk-level metrics across the four data-preparation versions (without CTC, $n = 72$). Hit@1 is document-level; nDCG, WRS, and Chain-MRR are chunk-level. Hit@1 and nDCG rise monotonically while WRS stays flat, the paradox examined in RQ2–RQ3.

5.2 Effects of Structural Chunking

> ***RQ2.*** *Does structural chunking help and does this fragmentation make it harder to find evidence?*

Primary metrics (Table 2): nDCG, WRS, WRec, Chain-MRR.

In our first data preparation stage, we move from naive (V1) to structural chunking (V2), which improves ranking quality but it fragments the context into smaller chunks (as described in Section 4.3.2). V2 has the largest single-step ranking impact of the study (nDCG +0.103, Figure 5), but Chain-MRR drops significantly (−0.064), while the marginal WRS decline (−0.035) is not significant.

To determine whether the decline of these metrics is an inevitable consequence of smaller chunk sizes or a real effect of structural chunking, the granularity control baseline V1-215 (Section 4.3.3) is examined.

When matching granularity, V1-215 performs worse than *both* V1 and V2 on every chunk-level metric (Figure 4). Comparing V2 directly to this baseline takes chunk size out of the comparison: reducing the window blindly (V1 to V1-215) lowers nDCG from 0.412 to 0.372, whereas the structural preparation at that same granularity reaches 0.526, +0.154 nDCG points over the control. Smaller chunks alone therefore do not help, and the V2 gain is not a size effect.

This control can not show which specific decision actually drove the gains (Section 4.3.2): heading-aware boundaries, atomic code blocks, the 20 % overlap between neighbours, the section path prepended as a header, and the field-structured task/solution parsing of the YAML exercises all change at this one step. What follows is therefore the effect of that group of decisions, not of the heading boundaries alone. It must be noted that V1-215 carries 17 queries whose evidence a 215-word window cuts apart (against four in V1 and none in V2), so part of its poor performance is lost evidence rather than lost ranking

quality, and the chunk-level nDCG itself favours finer chunking through its achievable maximum (Section 4.6.1). Section 6.5 discusses both.

The mechanism responsible for V2's inconsistent performance is evidence distribution, caused by finer fragmentation, rather than a lack of retrieval capability. Because V1 uses larger, 500-word windows, independent pieces of information are bundled into fewer chunks. V2 cuts the context into smaller pieces (median 215 words). In effect, this cleaner separation means that the exact same set of relevant facts is now distributed across a larger total number of chunks, naturally "lowering" the ranking. This effect explains why V2's performance seems worse at first, though it retrieves more evidence overall (proportion of gold references found 63.4 → 65.9 %; Figure 6). Because the evidence is spread over more individual pieces, V2's mean rank rises from 5.13 to 5.97 (worse), and its rank-CDF curve is below V1's from the beginning, only surpassing it from rank 13 on. It also causes the Chain-MRR computed for multi-hop queries to drop. The gap between the first relevant chunk and the last required piece widens: the median difference between Chunk-MRR and Chain-MRR rises from 0.37 (V1) to 0.50 (V2) over the 50 complementary queries. (Figure 5).

This bottleneck motivates the next pipeline stage: RQ3 investigates whether targeted keyword injection can lift the fragmented evidence back to the top of the ranking.

RQ2 — Is the performance drop because of smaller chunk sizes or structure?

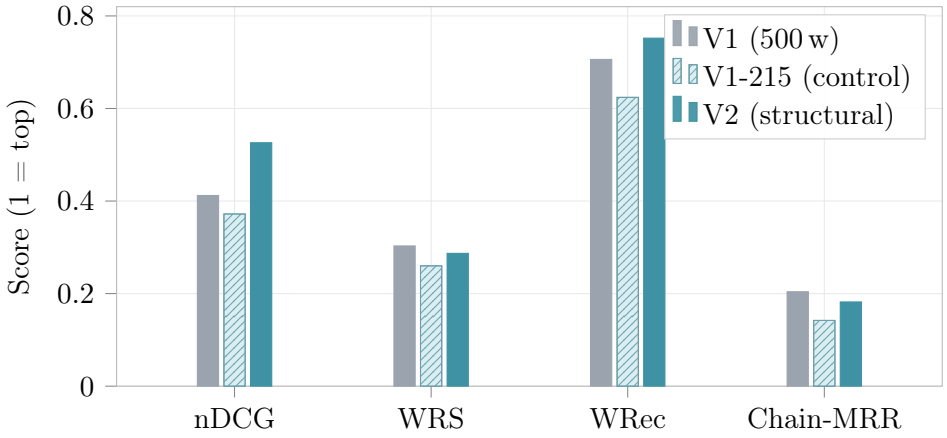

Figure 4: Control for granularity (without CTC, $n = 72$; levels, so the 17 queries whose evidence a 215-word window cuts apart are included at the score they achieve, Section 4.5). V1-215 applies V1's structure-blind fixed-size splitting at V2's median chunk length (215 words) to separate chunk size from the rest of the V2 preparation. It underperforms *both* configurations on all metrics, indicating that the V2 gain does not come from the smaller chunks; part of V1-215's poor performance is lost evidence rather than a ranking effect (Section 6.5).

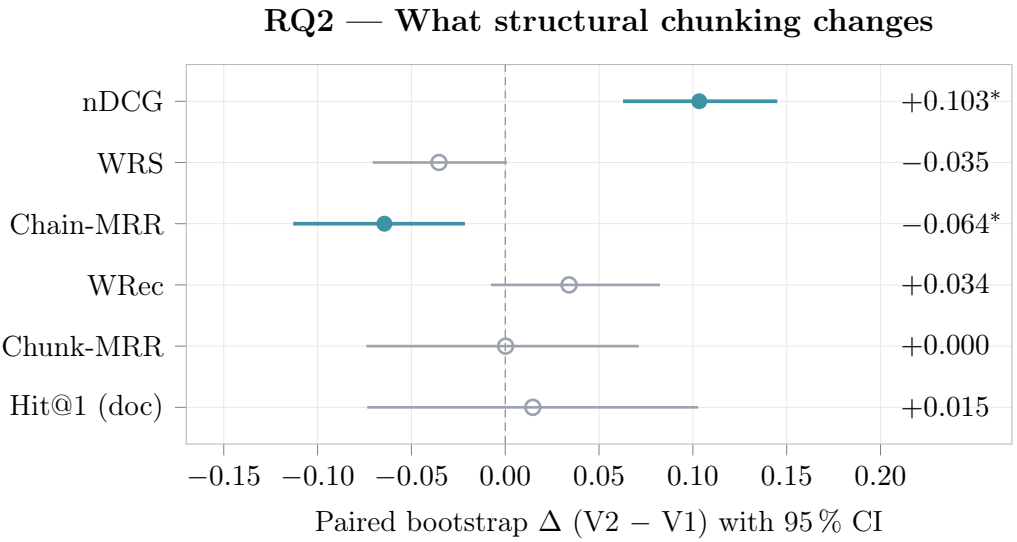

Figure 5: Paired step-gains from V1 to V2 (without CTC, $n = 68$ after excluding the four queries whose evidence is infeasible in V1; Chain-MRR on the $n = 62$ of these that carry at least one G_{high} reference). Filled markers indicate significance (95 % bootstrap CI excludes 0). Structural chunking significantly improves ranking quality (nDCG) but deepens the multi-hop bottleneck (Chain-MRR).

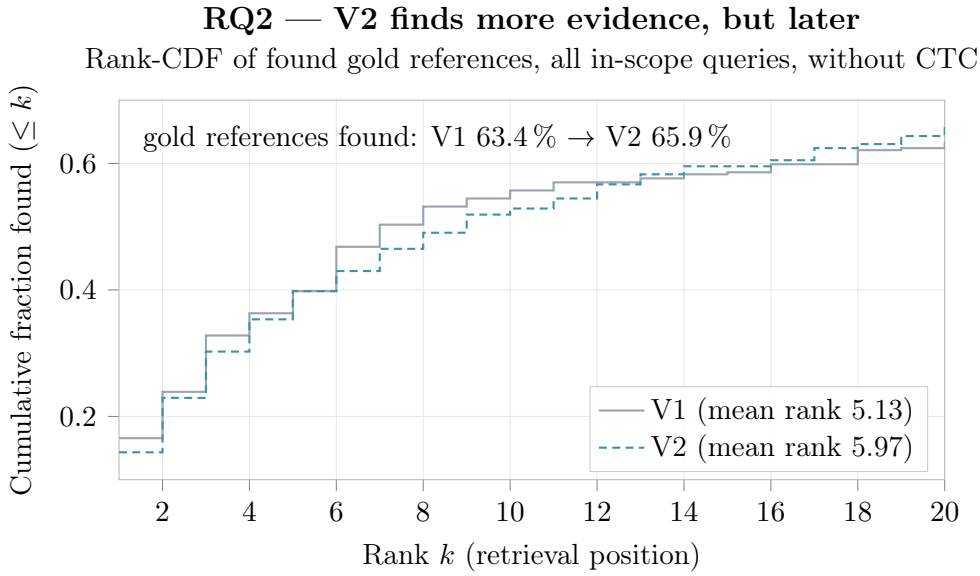

Figure 6: Rank-CDF of found gold references (all in-scope queries, without CTC). V2 retrieves more evidence overall (63.4 % → 65.9 %) but at higher (worse) ranks, surpassing V1 only from rank 13 on. This is why WRec rises while WRS does not: the two metrics share the same weighting and differ only in the rank-decay term (Section 4.6.1), so their divergence isolates *when* evidence is retrieved from *whether* it is retrieved.

5.3 Statistical vs. Generative Keyword Injection

RQ3. *Does keyword injection help, and does it matter whether keywords are extracted statistically (KeyBERT) or generated by an LLM?*

Primary metrics (Table 2): Hit@1, nDCG, WRS, WRec, Chain-MRR, Chunk-MRR.

When keywords are injected into the chunks before embedding, retrieval quality only improves for the LLM-generated variant. As shown in Figure 7, KeyBERT headers (V3a) show no significant gain on any metric and *significantly reduce* Weighted Recall (-0.019, CI $[-0.040, -0.002]$). LLM headers (V3b) deliver the only significant document-level gain of this preparation step (Hit@1 $+0.083$ to 0.681) and recover the V2 multi-hop drop (Chain-MRR $+0.035$, significant).[4] On WRS the LLM advantage is only marginal: of the 72 in-scope queries, 30 improve, 27 get worse and 15 stay exactly the same, five of the latter at WRS $= 0$ in both versions and therefore a shared failure rather than a shared success. The mean moves from 0.287 to 0.300. The reason for V3a's lack of significant gain and its significant reduction in weighted recall is analysed in Section 6.3, where the inspection of generated keywords reveals systematic, corpus-specific extraction failures.

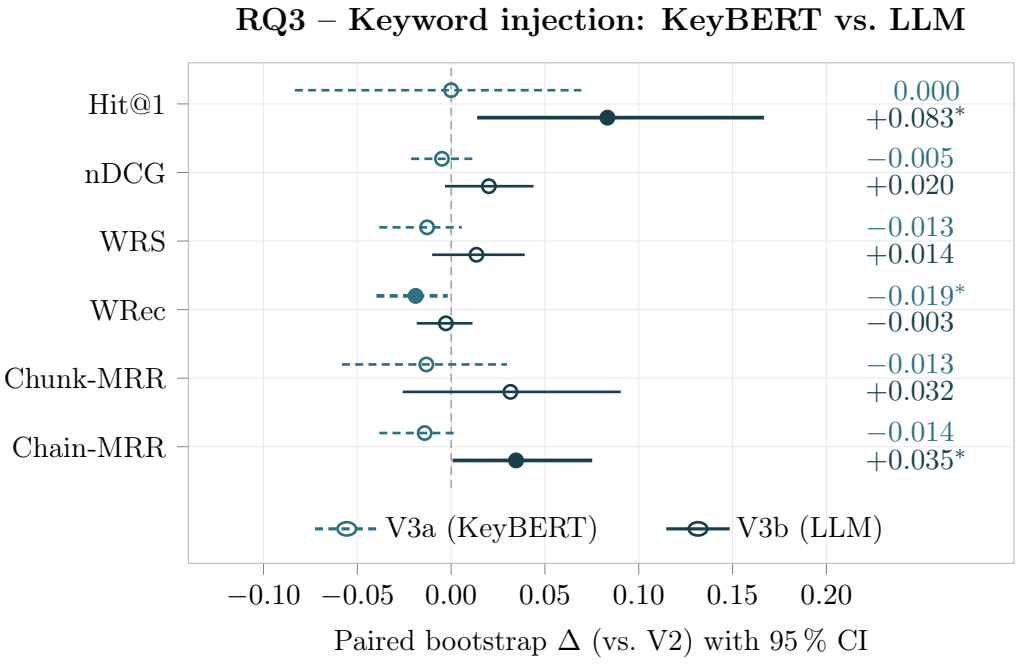

Figure 7: Paired step-gains from V2 to V3a and V2 to V3b for all metrics (without CTC, $n = 72$; Chain-MRR on the $n = 66$ queries that carry at least one G_{high} reference). V3a produces no significant gain. V3b significantly improves Hit@1 and Chain-MRR (the latter with a CI close to zero) while leaving WRS unchanged. Filled markers = significant (95 % bootstrap CI excludes 0), open = n.s.

[4] Chain-MRR is defined for every query carrying at least one G_{high} reference, and the paired comparisons use all $n = 66$ such queries so that the paired sample stays as large as possible. Restricted to the 50 complementary queries over which the Chain-MRR *levels* in Table 3 are averaged, the V2 → V3b gain has the same size ($+0.035$) but its interval touches zero ($[-0.001, 0.084]$), so this particular effect is better read as directional than as established. The V1 → V2 drop is significant under both definitions (-0.064 over 62 paired queries, -0.057 over the 46 complementary ones).

5.4 Impact of filling informational gaps

RQ4. *Does the hand-curated tutoring collection improve performance and does it benefit each version differently?*

Primary metric (Table 2): WRS.

The Curated Tutoring Collection improves performance in two ways and, as expected, the main impact is coverage. Eleven benchmark queries are out-of-scope (WRS = 0) without the CTC. Adding it makes them answerable, with the mean WRS reaching 0.78 in V1 and 0.91 in V2–V3b (Figure 8A). The gain from V1 to V2 is another indication of the benefit of structural chunking: V1 stores the whole CTC as one unstructured block, whereas V2 splits it into 19 separately retrievable articles. The granularity control V1-215 shares V2's index size but keeps V1's naive YAML handling and stays at V1's level (0.786 against 0.784), so the jump comes from splitting the CTC into 19 individually retrievable articles and not from finer chunks. Structural chunking benefits the curated collection just as it does the rest of the corpus. For the 72 queries already answerable without the CTC, the effect is a small gain across versions (WRS +0.026 to +0.034, Figure 8B), but only V1's gain is statistically significant, and with $n = 72$ the benchmark is underpowered to confirm an effect this size per version. Their confidence intervals overlap entirely, so there is no evidence that the CTC benefits one version more than another. It rather acts as an additional source to widen what the system can answer at all (Figure 8A). Whether these newly answerable queries can also be reliably distinguished from genuinely unanswerable ones is deferred to RQ7. Before that, RQ5 asks which queries carry the gains and which carry the fragmentation cost, and RQ6 asks whether that answer depends on the source material.

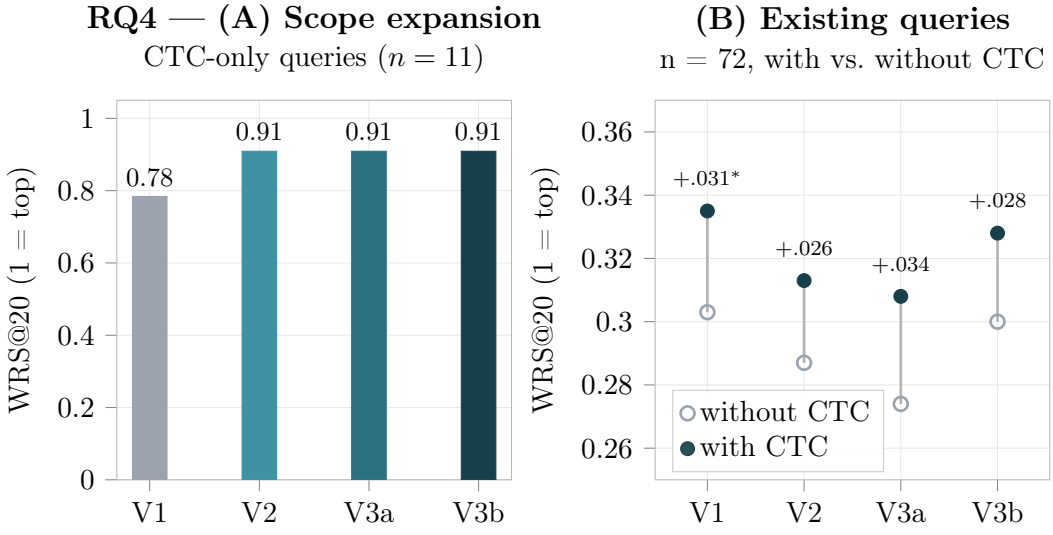

Figure 8: Impact of adding the Curated Tutoring Collection to the source data.
(A) *How does the CTC affect out-of-scope queries ($n = 11$)?* Without the CTC, WRS is 0. With it, mean WRS reaches 0.78 (V1) to 0.91 (V2–V3b). V2+ benefits from splitting the CTC into 19 structural articles.
(B) *How does the CTC affect already answerable queries ($n = 72$)?* The additional source provides a small, version-independent gain across all versions (open markers = without CTC, filled = with CTC. * = significant, 95 % bootstrap CI excludes 0). Both conditions are averaged over the same 72 queries, so the with-CTC values here are not the $n = 83$ ones of Table 7, which also average in the eleven CTC-only queries of panel (A).

5.5 Impact of Query Type

RQ5. *Does the benefit depend on query type (redundant vs. complementary)?*

Primary metrics (Table 2): Chunk-MRR, Chain-MRR.

A small difference is visible when comparing the two query types. Because redundant queries require only one adequate source (Section 4.5), Chunk-MRR represents the most fitting metric here and it rises steadily with every data-preparation stage: 0.410 → 0.412 → 0.439 → 0.486 (V1→V2→V3a→V3b), aggregating V1→V3b +0.076. Complementary queries show no such trend on Chunk-MRR: the level is mostly constant (0.627 → 0.643 → 0.617 → 0.661), including the same V3a dip already attributed to the KeyBERT regression in RQ3.

The complementary-specific effect is the multi-hop bottleneck shown by Chain-MRR (Figure 9B), which applies only to complementary queries: it falls from V1 to V2 (0.204 → 0.182) as structural chunking distributes evidence across more chunks (RQ2), dips further at V3a and recovers only with injected LLM keywords in V3b (0.218). While the distribution of costs across *queries* is now clear, RQ6 addresses the same question for the *material*: specifically, whether benefits and costs are spread evenly across both source-document types.

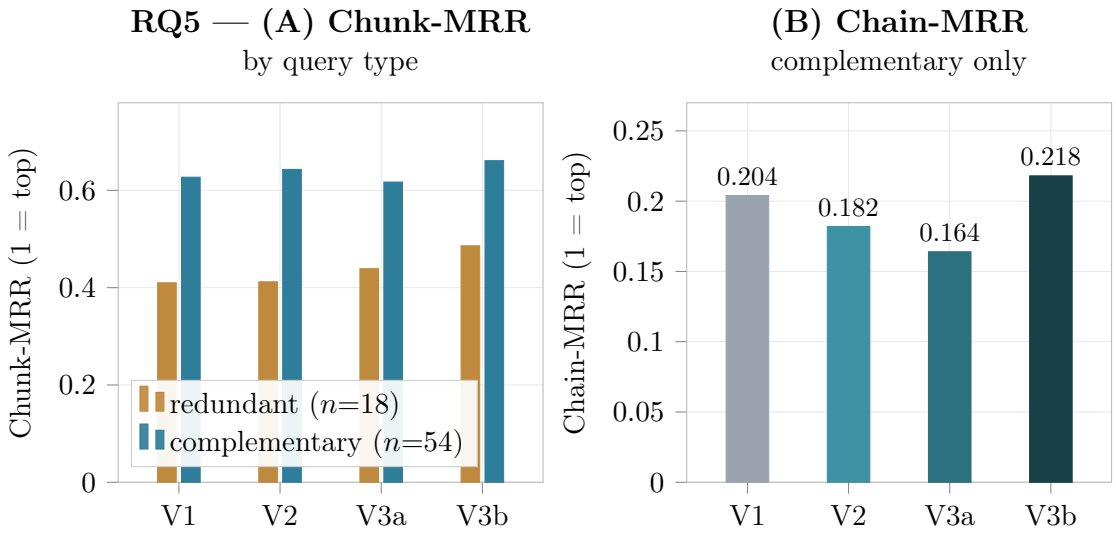

Figure 9: Impact by query type (without CTC).
(A) Chunk-MRR for redundant ($n = 18$) vs. complementary ($n = 54$) queries: redundant queries rise consistently at every stage (0.410 → 0.486), complementary queries stay similar (0.627 → 0.661, including a small V3a KeyBERT dip from RQ3).
(B) Chain-MRR on the $n_{\text{appl}} = 50$ complementary queries for which it is defined, captures the V1→V2 drop because of evidence distribution (RQ2) and its recovery with LLM keywords (V3b).

5.6 Influence of Source-Document Types

RQ6. *Are the improvements consistent across source-document types (YAML exercises vs. PDF slides)?*

Primary metrics (Table 2): nDCG, WRS.

The answer depends on which side of the RQ2 trade-off one looks at: the *benefit* of structural chunking is consistent across source types, but its *cost* is not. nDCG rises for all three document types (Figure 10, right) by a similar amount: PDF +0.107, YAML +0.091, mixed +0.106 (all 95 %-CIs exclude 0), so the core ranking-quality gain from RQ2 does not depend on the source type. WRS, however, drops most in YAML (−0.073) and only marginally in PDF (−0.032) and mixed (−0.024), reproducing the find-more-but-later pattern of RQ2 but concentrating it in YAML. Unlike the nDCG gains, none of these three intervals excludes zero, so the ordering is directional rather than individually established. What supports reading it as a real difference is the mechanism behind it: the V2 design splits each SmartBeans exercise into separate task and solution chunks (Section 4.3), so more of its evidence becomes individually findable, but spread across ranks.

Queries answered exclusively from YAML exercises remain the weakest segment in every configuration: in V3b they reach a document-level Hit@1 of 0.25 against 0.72 for PDF-only and 0.80 for mixed queries, and a chunk-level nDCG of 0.348 against 0.628 and 0.556. The Hit@1 value of 0.25 is in fact identical in V1, V2 and V3b (V3a: 0.333), so none of the three preparation stages moved the top-ranked document for this subset at all. What data preparation does improve is the ranking *within* the exercise material once it has been reached (nDCG 0.248 → 0.348 from V1 to V3b) without making an exercise file the first result for a query that needs one. With $n = 12$ this subset is small and the individual values are correspondingly unstable, but the gap to the other two source types is larger than any effect reported in RQ1–RQ3, and Section 6.4 returns to what it implies for the deployment.

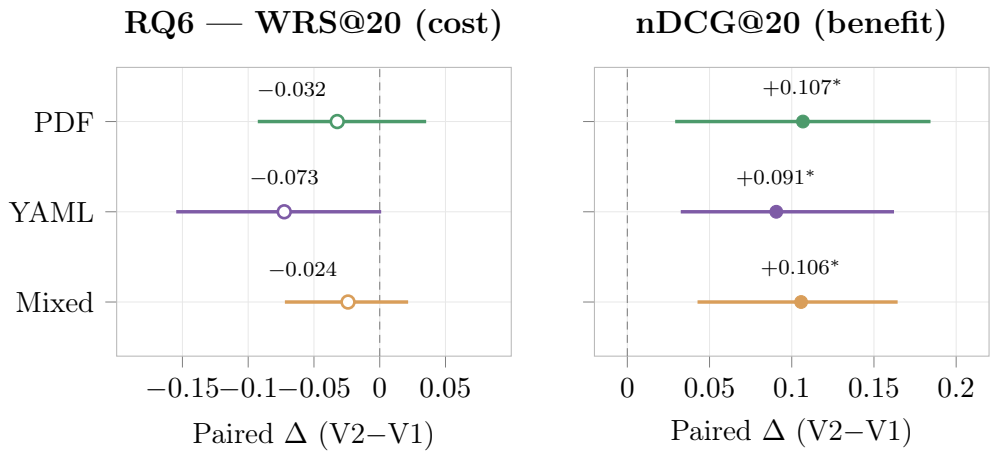

Figure 10: V1→V2 paired Δ by source-document type (feasible queries, without CTC). Left: WRS@20 (the RQ2 fragmentation cost); right: nDCG@20 (the RQ2 ranking-quality benefit). 95 %-bootstrap CIs, filled marker and * = significant (CI excludes 0). nDCG rises for all three types, WRS falls most for YAML exercises, which V2 splits into separate task and solution chunks, though no WRS interval excludes zero. The benefit of the structural preparation is universal, its dispersion cost is not.

5.7 Detection of Unanswerable Queries

RQ7. *Can the system distinguish answerable from unanswerable queries based on its retrieval scores alone?*

Primary metric (Table 2): AUROC.

On the small OOS sample, all four versions separate the 72 in-scope from the 11 out-of-scope queries reliably, with an AUROC of ≥ 0.907, peaking at 0.928 for V3b (Figure 11). Every curve is much above the chance diagonal. The separation is substantial in raw retrieval scores (`best_score`, defined in Section 4.6.2): in-scope queries have a median best_score of 0.48–0.51 versus an OOS median of 0.31–0.36 across the four versions, a per-version gap of 0.15–0.18. This indicates that the retrieval scores alone can serve as a signal for out-of-scope detection. Given the small OOS sample (n=11), these values are illustrative rather than definitive. What the preparation stages behind it cost at query time is the remaining deployment question.

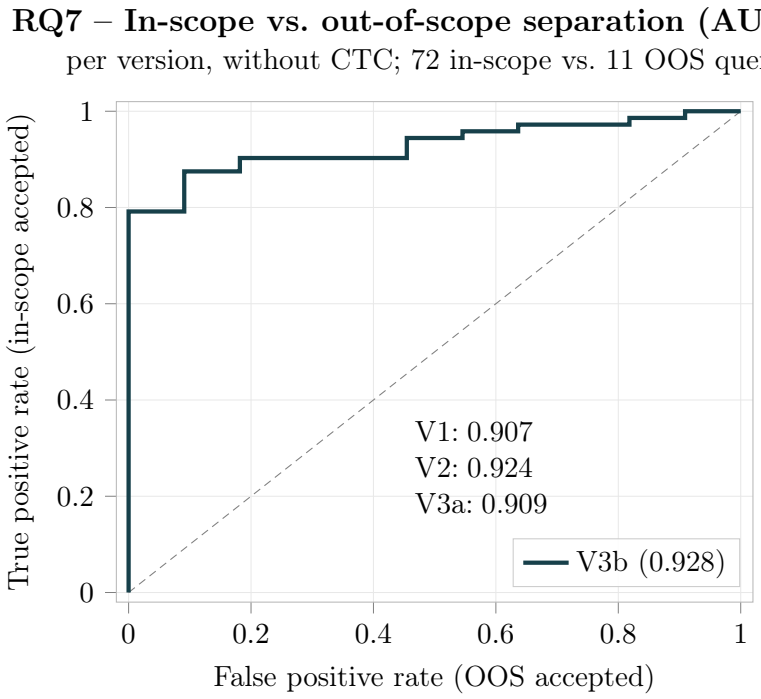

Figure 11: AUROC for separating in-scope ($n = 72$) from out-of-scope ($n = 11$) queries by best retrieval score, per version (without CTC). All curves are far above the chance diagonal. AUROC: V1 = 0.907, V2 = 0.924, V3a = 0.909, V3b = 0.928 (best of the four pipeline versions). A higher curve means unanswerable queries can be rejected at a lower cost in wrongly rejected answerable queries.

5.8 Quality vs. latency

RQ8. *What is the trade-off between retrieval quality and query latency across the versions?*

Primary metrics (Table 2): Latency (embedding / local / end-to-end).

Query latency is independent of data preparation, so there is no quality–latency trade-off at query time. Local retrieval (cosine + BM25 + RRF) stays in the same order of magnitude in every version, between 14 and 20 ms (V1: 17.1, V2: 19.7, V3a: 19.1, V3b: 14.4 ms). The spread between these four numbers is not a result. They are single-run means measured on a shared machine, the differences lie in the low milliseconds, and no ranking of the versions should be read into them: V3b indexes exactly the same 451 chunks as V2 and V3a and differs from them only by a few words of header text per chunk, so no mechanism exists by which it could genuinely be the fastest of the three. What the numbers do establish is the absence of a cost: neither structural chunking nor keyword injection slows local retrieval by an amount this setup could measure.

End-to-end latency (mean 3.3–4.3 s), however, is mostly slowed down by a single remote embedding API call, with worst-case response times taking 36 to 45 seconds. This is how the embedding model is managed in our current infrastructure. Since each query is embedded exactly once regardless of how the corpus was prepared, the differences between versions are API noise. The primary latency bottleneck lies in our embedding model management, not in the data preparation.

The only version-specific cost is at index-building time, not query time: V3b's LLM keyword generation annotated all 451 chunks in ≈419k tokens (≈397k prompt + 22k completion, `qwen3-30b-a3b-instruct-2507`), while V3a's KeyBERT step runs locally. Both are built once, are fully cached and reused for every query, which is why neither becomes part of the query latency.

6 Discussion

This chapter interprets the findings of Section 5 and puts them in the context of the research questions and the prior work. Section 6.1 explains the central paradox of this thesis (RQ1, Section 5.1): the improvement of document-level and ranking metrics, while WRS stays flat and resolves it using the granularity-matched control (RQ2) and the document-type comparison (RQ6). Section 6.2 discusses the trade-off in structural chunking (RQ2), confirmed by the query-type analysis (RQ5), Section 6.3 studies why statistical and generative keyword injection have such different effects (RQ3, Section 5.3). Section 6.4 points out practical implications for the Arcana deployment on which a tutoring system for the course would build: the role of the Curated Tutoring Collection in widening coverage (RQ4), out-of-scope detection (RQ7), and the cost–latency trade-off (RQ8). Section 6.5 discusses the limitations on which these interpretations depend.

6.1 The flat WRS paradox

Across the four pipeline versions, data preparation alone raised the chunk-level nDCG (How good is the overall ranking of the relevant chunks?) from 0.412 to 0.546, but WRS (How early are the score-weighted evidence chunks retrieved?) remained essentially unchanged ($0.303 \rightarrow 0.300$, RQ1). There are two explanations for this difference sharing the same root (the smaller chunks introduced in V2), but one is an effect of the changed data and one is a limitation of the metrics themselves.

Evidence distribution. The first one is a structure effect. As shown in RQ2 (Section 5.2), structural chunking distributes the same set of relevant facts across a larger number of smaller chunks. Two complementary views on recall make this visible. Counted per evidence piece and unweighted by the graded matching score, V2 places more of the 315 gold references (without CTC) into the top-20 than V1 ($63.4 \rightarrow 65.9\,\%$). Averaged per query and weighted by the matching score, WRec rises from 0.706 to 0.752.[5] Besides this, in V2 four queries become answerable whose gold references are not locatable in the V1 index at all (Section 5.2), so structural chunking widens what is retrievable in principle as well as what is retrieved in practice. The individual pieces, however, appear later in the ranking and for multi-hop queries the gap between the first and the last required piece widens.

The improvements therefore appear most clearly for nDCG and WRec. WRec rises because more evidence is found at all. nDCG rises for two reasons. First, its $\log_2$ discount is less strict at deeper ranks than at the first ranks, so whether an evidence is found at rank 9 or rank 15 barely changes the score. Second, its ideal ranking, which the finer index makes achievable where V1's co-located references had limited it, rises with granularity as well (Section 4.6.1, Section 6.5). WRS, however, penalises shallow and deeper ranks in the same way with a reciprocal decay (1/rank): the score drops by a constant relative factor across all ranks (doubling the rank always halves the score). WRS therefore only rewards finding *all* evidence pieces early at once, which makes it structurally harder when the index is finer and the evidence is distributed over more chunks.

[5] The two figures are not interchangeable: the percentage is a micro average over all 315 individual gold references, each counted once regardless of its graded score, whereas WRec is a macro average over the 72 in-scope queries of the per-query score-weighted recall. A query with many low-weight references therefore contributes no more than one with a single decisive reference.

Confirmation across source-document types. RQ6 (Section 5.6) confirms this mechanism. The nDCG gain from structural chunking is steady across document types (PDF +0.107, mixed +0.106, YAML +0.091; all CIs exclude zero), so the ranking-quality benefit does not depend on the source. The WRS cost, however, appears where V2 fragments the most: YAML exercises, which V2 splits into separate task and solution chunks (Section 4.3), lose −0.073 WRS, compared to −0.032 for PDF and −0.024 for mixed queries. All three of these intervals include zero (RQ6), so the ordering is a directional observation rather than a per-type effect established on its own. Read as such, it follows the pattern the evidence-distribution section above describes: the more a source is split, the more its evidence is distributed across ranks, and the more WRS drops. Whether a less aggressive, YAML-specific chunking strategy is needed depends on the open question discussed in the following Section 6.2: whether rank position within the context actually affects generation quality.

Rank-based metrics are not granularity-robust. The second finding is methodological. Chunking experiments necessarily change the size of the index: in this study, the corpus grows between versions (254 chunks in V1 to 432 in V2/V3a/V3b in the without-CTC condition, +70 %; 255 to 451 once the CTC is included, Table 1), because the shift from a fixed 500-word window to the heading-aware structural chunking splits the same source text into more, smaller sections. WRS and Chunk-/Chain-MRR all use a raw 1/rank decay, which does not account for how many competing chunks exist in the index.

If an evidence passage stays at a constant relative rank (e.g., top 2 %), but the corpus grows from 254 to 432 chunks, its absolute rank naturally shifts from ≈5 to ≈9. This shift alone lowers the 1/rank metrics far more than $\log_2$ metrics. This difference in metric definition therefore explains why nDCG rises while WRS, Chunk-MRR, and Chain-MRR stay flat or fall when the corpus is growing, independently of any change in evidence distribution (Section 5.2). A granularity-robust variant would decay with the rank *percentile* ($\text{rank}/N_{\text{chunks}}$) instead of the absolute rank. Computing it retrospectively would require the full ranked list per query rather than a top-k cutoff (Section 6.5).

WRS is not annotation-density-robust either. Index size is not the only unnormalised quantity in the WRS definition. Because the metric averages the rank decay over *all* references of a query (Section 4.6.1), each additional annotated source lowers the score that is reachable at all, even when retrieval behaves identically, and the benchmark makes this visible: WRS correlates negatively with the reference count in every version while Chunk-MRR, which carries no such term, does not (Appendix C). Both attributes follow from the way the metric normalises neither for the number of chunks competing for a rank nor for the number of references competing for the weighted mass. Both affect the absolute WRS level of this study rather than the comparison between versions, since the annotation is identical in every configuration.

The granularity-matched control resolves the paradox. The V1-215 control introduced in RQ2 separates the two effects: its index is nearly the same size as V2's (Table 1), so the comparison V1-215 → V2 is close to size-neutral. What remains of the difference leaves V1-215 with the *larger* index, so the corpus-size effect described above works against V2 and cannot be what produces its advantage. In this comparison *all four* chunk-level metrics improve, the two rank-decaying ones (WRS, Chain-MRR) as clearly as the two that had risen all along (nDCG, WRec); Figure 4 shows the levels. This pattern

indicates that structural chunking improves evidence depth as well as ranking quality once index size is held constant: the flat V1→V2 WRS is more a measurement artefact than a real retrieval failure. The methodological suggestion for future work is including an evaluation step whenever a preparation step changes chunk granularity and it should be paired with a granularity-matched control, so that rank-based metrics are compared at equal index size.

Taken together, the paradox of RQ1 is itself a result. If this thesis had evaluated only document-level metrics or nDCG, the granularity dependence of chunk-level rank-based metrics would not have been noticed. Revealing this paradox required the score-weighted chunk-level metrics and resolving it required the granularity-matched control.

6.2 The Multi-Hop Bottleneck

The granularity control (Section 4.3.3) shows that the V2 ranking gain is not a chunk-size effect. Adjusting V1's windows to V2's median chunk length (≈215 words) lowers every chunk-level metric, whereas the structural preparation at that same granularity surpasses the baseline (+0.154 nDCG over V1-215). What the gain can be attributed to is that preparation as a whole (heading-aware boundaries, atomic code blocks, the 20 % overlap, the section-path header and the field-structured YAML parsing, Section 4.3.2) since all of them change at the same step. The design separates this bundle of decisions from chunk size, not one from another. At the same time, structural chunking deepens the multi-hop bottleneck (paired Chain-MRR −0.064 over the 62 paired queries carrying a G_{high} reference, significant, RQ2). We can exclude two explanations for this bottleneck:

Chunk size. The first one addresses the architectural question: should V2's chunk size be increased to prevent evidence distribution? However, structural chunking follows the document structure. Artificially expanding the window would force unrelated information into the same chunk (Section 4.3.2).

The fixed retrieval depth. The second is the suspicion that $k = 20$, held constant across versions (Section 4.1), becomes a disadvantage for the finer-chunked versions, whose top-20 covers less text and could lose more evidence. An inspection of the stored top-20 lists shows that this is not the case: a must-have reference is missing from the top-20 in *fewer* queries in V2 than in V1 (22 against 25 of the 66 queries carrying one), so V2's evidence is not cut off more often and $k = 20$ is not the source of the Chain-MRR drop.

The spreading of evidence across ranks. This is what is actually responsible for the lower scores. In V1, several gold references are inside the same 500-word chunk, so the first found piece and the last required piece share one rank more often: the median rank spread between them is 3 ranks (mean 4.4, over the queries with at least two retrieved G_{high} references). Structural chunking separates these references into different chunks and the median rank spread increases to 5 ranks (mean 6.2). Since Chain-MRR scores the *last* required piece ($1/\max_i \text{rank}_i$), this spread lowers the metric even when every piece is found. The multi-hop cost of structural chunking is therefore a *rank-spread* effect within the retrieved list, not an effect of lost evidence: V2 delivers the complete evidence set to the top-20 more often than V1, but distributes it across more ranks which leads to a lower WRS score (Section 6.1).

Implications for a deployed tutoring system. This distinction matters in deployment: the generator receives the full top-k list as context, so what matters is whether the evidence is *included* in the retrieved set. In this regard V2 improves over V1 (WRec 0.706 → 0.752, fewer incomplete chains). The multi-hop cost is therefore mostly an evaluation artefact, not necessarily a limitation of what the generator would actually receive as context. However, this spread should not be dismissed entirely: evidence position can affect generation quality [27], meaning a wider rank spread may still lower the generation performance that this retrieval-only evaluation cannot observe (Section 6.5). Within the design of this thesis where the retriever is held constant, we attempt to recover the falling scores by keyword injection (RQ3) since corrections on the retrieval configuration (such as merging neighbouring chunks after retrieval, expanding a hit to its parent section, or the context-length-matched k discussed above) are outside the scope of the data-centric design (Section 2.4).

Confirmation across query types. RQ5 (Section 5.5) shows which queries are affected by this bottleneck. Redundant queries, answerable from a single source, are unaffected by the distribution of evidence and simply improve at every stage (Chunk-MRR 0.410 → 0.412 → 0.439 → 0.486 from V1 to V3b). Complementary (multi-hop) queries behave differently on Chain-MRR, the metric that scores the deepest required piece: it drops exactly at the V1→V2 shift where evidence is distributed across more chunks (0.204 → 0.182), falls further with the noisy KeyBERT headers of V3a (0.164), and recovers only with the vocabulary-bridging LLM keywords of V3b (0.218), which aligns with the findings discussed above. The multi-hop bottleneck is therefore not a general property of the pipeline but a cost that falls entirely on the subset of queries that need several evidence pieces for a grounded answer.

6.3 Why Statistical Extraction Fails

Keyword injection produced a very clear contrast: KeyBERT headers (V3a) had no significant gain and even reduced weighted recall, whereas LLM headers (V3b) delivered the only significant document-level gain of the pipeline and also recovered the drop in multi-hop metrics of V2. Inspection of the injected headers suggests three reasons for why there is no significant gain for V3a on this corpus.

How KeyBERT acts on this corpus. An inspection of the injected headers puts a number on the problem: not quite one in five of the keywords V3a writes into the 451 chunks contains markup or code characters, against one keyword in eighty for the LLM (Table 5, Appendix D). Three mechanisms generate them. First, KeyBERT often re-extracts the injected structural headers themselves: since every V2 chunk is prefixed with a heading line (e.g. `# Kapitel 11 > 11.4 Felder`), tokens such as `# kapitel` or `# anhang` are extracted as keywords. `# kapitel` alone occupies 104 of the 3179 keyword slots and is the most frequent V3a keyword of all, so the enrichment step spends part of its budget re-describing the structural context that V2 had already added. Second, KeyBERT ranks terms by corpus-wide distinctiveness, which causes the OCR artefacts introduced by Docling's PDF-to-Markdown conversion to score as highly unique. Rare noise fragments such as `}}}_return` or `printf("%ld\n")` are therefore chosen over semantically meaningful keywords. Third, short tokens such as `cd` are filtered by KeyBERT's minimum character threshold, although they are critical for the query. Instead, longer paraphrases

(*working directory*, *Arbeitsverzeichnis*) are used that miss exact-term queries. Together these artefacts distort the chunk embeddings more than the additional vocabulary helps.

The POS filter introduced in Section 4.3 does not prevent any of this, and the reason is worth stating precisely, because it is not a property of KeyBERT. Extracted phrases such as `# define mwst 15.0 # define mwst\_erm 7.0` span seven tokens and could not have been produced by the n-gram fallback, which is capped at three, so the `KeyphraseCountVectorizer` was active and did apply its pattern. The German `de_core_news_sm` model, however, annotates tokens such as `#`, `include` and `stdio.h` as nouns, so the resulting sequences satisfy the noun-phrase pattern and pass the filter as grammatical candidates. What fails here is the application of a general-purpose German news tagger to a corpus that consists in large part of C source code and converted slide markup. Appendix D contrasts both keyword sets on three chunks.

Why generative extraction succeeds. The LLM keywords are generated differently: the model generates vocabulary from meaning, not statistics, and is not affected by OCR artefacts or token-length constraints. The keywords it produces bridge the vocabulary gap between colloquial student queries and instructional text, which is why the gain is mainly at the document level: routing the query to the correct source requires only one strongly matching chunk, whereas WRS improvement requires all evidence chunks to rank early at the same time, which vocabulary bridging alone cannot solve. Taken together, V3a's failure is not necessarily due to statistical keyword extraction itself, but to corpus-frequency distinctiveness operating on noisy text, combined with the method and pipeline design limitations described above. Yepes et al. [41] are, to our knowledge, the closest prior use of LLM-generated keywords as chunk metadata in a RAG pipeline, but their design does not permit the comparison made here: their keyword variant is measured against fixed-size baselines rather than against the same chunks without keywords, and it raises their question-answering accuracy while lowering their page-level retrieval accuracy (Section 3.2.3). Because V2, V3a and V3b share the same 451 chunks position for position and differ only in the injected header (Section 4.3), the effect measured in this thesis is attributable to the keywords themselves rather than to a co-varying change in chunking. On this corpus that effect is positive but narrow: it is significant at the document level (Hit@1 +0.083) and on the multi-hop metric (Chain-MRR +0.035), remains within the noise on WRS (+0.014, CI includes zero), and is bought with a one-off index-time cost (≈419k tokens, Section 5.8) that statistical extractors avoid. Whether this direction transfers to other corpora and languages is not established by a single study (Section 6.5).

6.4 Implications for Deployment

For the Arcana deployment, the recommendation is to adopt the idea of V3b's ingest configuration: structural chunking with hierarchy embedding, LLM-generated keyword headers, and the Curated Tutoring Collection as a dynamic corpus extension. Every component of this recommendation is a data-preparation decision while other configurations such as the embedding model, the hybrid retriever and the index remained untouched, which is the paradigm of DCAI [42, 17].

How firm the V3b recommendation is. V3b is the strongest configuration overall, but not on every measure, and the recommendation should be read with that qualification. Its advantage is concentrated where routing and ranking are measured: Hit@1, nDCG,

Chain-MRR and AUROC all place it first (Table 3, RQ7). On coverage it does not improve on V2 at all, Recall@20 and WRec are marginally lower, Strict@20 is identical, and Hit@5 is in fact the *lowest* of the four versions (Table 7). What the keyword headers do is lift the right chunk to the top of a list that already contained it, rather than putting more evidence into the top-k that was not there before. For a deployment that hands the generator a whole top-k list, that is the smaller of the two possible benefits. V3b is therefore best justified by its strong Hit@1 and the multi-hop recovery, the two effects whose paired intervals exclude zero (RQ3). Of the coverage differences only WRec was tested, and it lies within the noise. V2 remains a defensible choice where LLM annotation is not available.

Coverage extension via the CTC. The Curated Tutoring Collection results (RQ4) confirm that such a curated collection widens coverage. Its benefit is making the unanswerable queries answerable whereas its effect on already answerable queries is small and version-independent. For deployment this suggests following the data-maintenance strategy described by Zha et al. [42]: In our case, tutors and course instructors should extend the collection whenever student questions require information beyond the course material, because closing a coverage gap is worth more than any ranking optimisation on existing content.

The exercise material is the weak point, not the script. The source-type breakdown (RQ6, Section 5.6) locates the next data-centric intervention more precisely than the aggregate metrics do. In every configuration the SmartBeans exercises are retrieved substantially worse than the lecture script, and no preparation stage closed the gap: in V3b, queries answered from exercises alone reach a Hit@1 of 0.25 against 0.72 for the script. This is plausible from the material itself. A task chunk consists of a title, a tag list and a problem statement written for a reader who already knows which exercise they are looking at, so it contains little of the explanatory prose that a student question can match against, and the solution chunk is C code. Structural chunking makes these fields individually retrievable, and keyword injection adds the terms an exercise *mentions*, but neither supplies what the exercise is *about*. Annotating the exercises with the concepts they practise, ideally by the course staff who already maintain the tag field, is therefore a more promising next step for the Arcana corpus than further work on the lecture script, which V3b already retrieves well.

Out-of-scope detection. The out-of-scope analysis (RQ7), defined in Section 4.6.2, indicates that retrieval scores alone separate answerable from unanswerable queries well (AUROC ≥ 0.907), which is useful in educational deployments, where both hallucinated and withheld answers count as failures [28] and a threshold trades one against the other.

The separation is measured under a relatively hard condition, since the OOS queries in this benchmark share the same topic as the corpus. The measured separation therefore indicates how separable in-domain queries are, not how separable obviously off-topic ones would be. However, as mentioned in Section 5.7, further validation on larger corpora is needed since this benchmark evaluated only eleven OOS queries.

Cost at query and index time. Finally, the cost analysis (RQ8) shows that these quality gains have no additional cost at query time: all preparation costs are paid once at index-building time and cached. The one exception is V3b's own preparation step: generating LLM keywords for all 451 chunks required ≈419k tokens (≈397k prompt + 22k

completion, `qwen3-30b-a3b-instruct-2507`). This cost is paid once, is fully cached, and recurs only if the corpus changes, which is a reasonable trade-off for the gained retrieval quality (RQ1, RQ3, RQ7). V3b therefore delivers its ranking gains at a query latency similar to the naive V1 baseline, at a modest, once generated and cacheable index-time cost.

The latency bottleneck. The real latency bottleneck, however, is not data preparation but the embedding API itself. End-to-end query latency is driven by a single remote embedding call (mean 3.3–4.3 s, worst case 36–45 s), and the same reliance on individual, per-chunk API requests also slows down index-building, since every chunk must be embedded before it becomes searchable. Because this bottleneck is identical across all four versions, no data-preparation strategy studied here can fix it.

6.5 Limitations

Rank-based metrics are not corpus-size normalised. WRS, Chunk-MRR, and Chain-MRR all decay as 1/rank, computed for indexes with different sizes across versions (254 chunks in V1 vs. 432 in V2/V3a/V3b in the without-CTC condition that all reported comparisons use; 255 vs. 451 with the CTC included, Table 1). None of these metrics have a correction for how many chunks are competing for a given rank. As Section 6.1 shows, the granularity-matched V1-215 control shows that part of the V1→V2 decrease in these metrics is this corpus-size effect rather than a real drop in retrieval quality, but the control variable only explains the *direction* of the effect. A corpus-size-normalised variant (e.g. decaying by rank percentile, rank/N_{chunks}, instead of absolute rank) would quantify this exactly, but requires the full ranked list per query rather than the top-20 cutoff stored by the current pipeline, so computing it in retrospect was not possible here. The same missing normalisation makes WRS sensitive to the number of references a query carries (Section 6.1); this affects the absolute level of the metric but not the comparison between versions, because the annotation is identical in every configuration.

Since this bias underestimates rather than exaggerates the benefits of smaller chunks, the decreasing WRS and Chain-MRR from V1→V2 should be read as the lower bound. The granularity-matched comparison in Section 6.1 indicates the direction that would be expected if the rank-based metrics were corpus-size normalised, but it cannot quantify the size of the correction.

nDCG is not granularity-neutral either, but in the opposite direction. The corpus-size effect above depresses the finer-chunked versions. The chunk-level nDCG defined in Section 4.6.1 carries a bias in the other direction, and it acts on the headline result of this study. Because rel(c) takes the maximum over the gold references co-located in a chunk while IDCG counts each of them, a query whose references sit in one chunk cannot achieve nDCG = 1.0: four must-have references inside a single 500-word chunk give an IDCG of 3.13 against a top DCG of 1.0, limiting that query at 0.32. The achievable maximum rises as chunks get finer, and V1 is precisely the version in which co-location is most frequent (Section 6.2), so part of the nDCG gain from V1 to V2 is a raised maximum rather than a better ranking. The size of that part is not quantified here.

Two observations argue against it being the dominant term, without settling the question. First, the granularity control shares V2's chunk size and therefore its raised maximum, yet V1-215 falls *below* V1 on nDCG (Figure 4). If this effect dominated, it

would have to sit at or above it. That argument is weakened by the 17 queries whose evidence V1-215 cannot locate at all (see below), which depress it independently. Second, and more directly, Recall@20 and Strict@20 measure only whether the must-have chunks are in the top-20 and involve no ideal ranking at all, so no achievability maximum applies to them and both improve from V1 to V2 (Table 3), which is a gain the co-location mechanism cannot produce. An nDCG variant whose IDCG is built from the relevance grades *as they are attainable under the version's own chunking* would remove the bias entirely; the evidence-coverage validation (Section 4.5) already establishes the reference-to-chunk mapping this requires, so the variant is computable from the existing caches and is left to future work.

Feasibility of the granularity-matched control. The V1-215 control shares V2's index size, but not its evidence coverage: for 17 of the 72 in-scope queries the gold references cannot be located in the V1-215 index at all, compared with four in V1 and none in V2. Part of the metric gap between V1-215 and V2 is therefore not a ranking effect but a consequence of fixed 215-word windows cutting evidence spans apart. This strengthens rather than weakens the argument of Section 6.1, since it shows that a naive reduction of window size degrades the corpus in a way structural chunking does not, but it means the control isolates index size only, not evidence locatability.

A second qualification concerns what "granularity-matched" means here. V1-215 and V2 agree on the size of the index but not on how that size is distributed across the two source types (Table 1). Because V1-215 inherits V1's naive YAML handling, its 215-word window applies only to the lecture script, which it splits into 357 chunks against V2's 241, while each exercise file stays a single block, leaving 97 chunks against V2's 210. The totals agree because two opposite deviations happen to cancel. For the argument of Section 6.1 this is not fatal, since V1-215 is the *finer* of the two configurations on exactly the material that carries most of the benchmark's evidence and still loses on every chunk-level metric, which makes the structural reading harder rather than easier to obtain. It does mean, however, that the control holds the number of competing chunks constant only in aggregate; separating size from structure *within* each source type would require a per-type control that this study does not provide.

V2 is a group of changes. The step from V1 to V2 changes several preparation decisions at once: heading-aware chunk boundaries, atomic code blocks, a 20 % overlap between neighbouring chunks, the section path prepended as a header, and the field-structured task/solution parsing of the YAML exercises with German field labels (Section 4.3.2). The granularity control isolates exactly one member of this bundle, chunk size. Every effect this thesis attributes to "structural chunking" is therefore the joint effect of the remaining variables, and each major finding can also have other plausible explanations: the overlap alone could produce part of the WRec gain, because it makes an evidence span locatable in two neighbouring chunks instead of one, and the section-path header alone adds vocabulary to the embedded text in the same way the V3 keyword headers later do. Establishing which decision carries the effect requires an ablation that varies them one at a time, which this study does not provide. The V2 $\rightarrow$ V3 comparison is not affected by this: V3a and V3b differ from V2 by the injected keyword header and nothing else (Section 4.3), so that step is a genuine single-factor comparison, although it is measured against a baseline that inherits the bundled V2 configuration.

Benchmark size and annotation. The benchmark consists of 83 queries with 342 gold references. The references and their graded matching scores were annotated by the author and reviewed by the course instructor. No second independent annotation is included, so a degree of annotator bias in the graded scores cannot be excluded. The sample is sufficient to detect the large effects reported in RQ1–RQ3, but not the smaller ones, as RQ4 showed: WRS gains of +0.026 to +0.034 were consistent in direction across all versions, but only one reached significance. Such marginal effects should be read as indicative rather than definitive. The OOS analysis depends on eleven queries that become out-of-scope once the CTC is excluded. The AUROC values are stable across versions, but no threshold calibration or rejection-rate estimate should be derived from a sample of this size.

Multiple comparisons. RQ2–RQ6 report 28 paired bootstrap intervals, none of them corrected for the number of comparisons made. At the 95 % level, one or two of these intervals would be expected to exclude zero purely by chance. While the nDCG gains from structural chunking (RQ2, confirmed per document type in RQ6) are large enough not to be affected by this, the marginal effects in RQ3 and RQ4 should be read as consistent directional trends rather than individually established findings.

Single course and model configuration. All results were obtained on one German-language C programming corpus with one embedding model (`Qwen3-Embedding-4B`), one keyword-generation model and prompt (hash-locked for reproducibility, Appendix E), one hybrid retrieval configuration, and one vector-database implementation (zVec, Section 4.1), whose retrieval behaviour is treated here as a fixed and correctly functioning component rather than as an object of evaluation. The direction of the findings (structure over size, generative over statistical enrichment, coverage over ranking) is potentially transferable, but the sizes of the metric gains are specific to this corpus and configuration. In particular, the reasons why KeyBERT fails, described in Section 6.3, depend on properties of this corpus that may not be present in cleaner ones.

The retrieval configuration is a precondition, not the deployed one. All experiments use hybrid dense + BM25 retrieval fused via RRF (Section 4.1), whereas Arcana currently retrieves on the dense vectors alone. This decision was made because hybrid retrieval is the stronger configuration in the literature (Section 3.2.1) and the one Arcana is intended to adopt. Experimenting with a hybrid retriever keeps a weak retriever from hiding the ingest effects this thesis intends to measure. The reported effect sizes therefore hold for a hybrid retriever and not necessarily for the deployment as it runs today.

Retrieval-only evaluation. This thesis measures only the retrieval component of RAG (Section 1). Correct retrieval is a necessary but not sufficient condition for a correct tutoring answer: whether the generator uses the retrieved evidence faithfully and whether students benefit pedagogically is not evaluated in this thesis. This also limits the interpretation in Section 6.2, where the wider rank spread of V2 is treated as largely harmless because the evidence still reaches the context window. Without a generation-side evaluation the relevance of the rank position cannot be verified. The chunk-level gains reported here define what a deployed system can build its answers on, not the quality of those answers.

7 Conclusion

LLM-based tutoring systems fail on course-specific, long-tail material that never appeared in their training corpus. RAG is the established answer to this problem, but a generator can never be more reliable than the chunks its retriever delivers. Most work on this bottleneck optimises models, retrievers, and hyperparameters while treating the corpus as fixed. This thesis examined the retrieval stage of an educational RAG system from a Data-Centric AI perspective: embedding model, hybrid retriever, and index configuration were held constant across all experiments and only the preparation of the data was varied: from naive fixed-size chunking (V1) via heading-aware structural chunking with hierarchy embedding (V2) to keyword injection, either statistically extracted (V3a) or LLM-generated (V3b). Measuring the effect of these stages required a benchmark that did not exist for this domain and was created within this work: 83 real student queries with 342 expert-annotated gold references at *chunk* level, evaluated with score-weighted metrics that measure retrieval performance on the passage actually needed for an answer rather than on the whole document. That benchmark turned out to be the precondition for everything reported below, since the findings are invisible at document level.

Central findings. The answer to the main research question is affirmative but differentiated. Data preparation alone, without changing the model or retrieval architecture, raised chunk-level nDCG from 0.412 to 0.546 (RQ1), while the two coverage metrics peaked one stage earlier, at V2. Part of the V1 $\rightarrow$ V2 share of that gain belongs to the metric rather than to the retrieval, because a coarse index in which gold references share a chunk cannot attain the ideal ranking at all; that the gain is not merely this ceiling is indicated by the coverage metrics, which involve no ideal ranking and improve at the same step (Section 6.5).

It also matters *how* enrichment vocabulary is produced: LLM-generated keywords delivered the only significant document-level gain of the pipeline (+0.083 Hit@1 over V2) and recovered the multi-hop bottleneck that structural chunking had developed, while statistical extraction failed on this corpus and left V2 as the stronger configuration on coverage (RQ3).

The evidence-depth metric WRS, however, stayed flat across all versions, and Chain-MRR even dropped at the V1 $\rightarrow$ V2 step. Resolving this paradox turned out to be a methodological finding. Structural chunking grows the corpus by some seventy per cent, and WRS, Chunk-MRR and Chain-MRR all decay as 1/rank without accounting for how many chunks compete for a given rank: a passage that keeps its *relative* position loses score simply because the index got finer. The granularity-matched control V1-215 was built to separate these two effects and it does: at equal index size *all four* chunk-level metrics improve in V2, the rank-decaying ones included (RQ2, Figure 4). Once the corpus size is held constant, structural chunking improves evidence depth just as it improves ranking quality, and the flat WRS of the headline comparison is mostly a measurement artefact of rank-based metrics rather than a retrieval failure.

Beyond the pipeline stages, the Curated Tutoring Collection proved to widen coverage rather than to increase the ranking quality: it makes eleven previously unanswerable queries answerable while its effect on already answerable queries is small and version-independent (RQ4). The preparation gains are consistent across query types and source-document types, with the cost of evidence distribution in V2 concentrated in the finely split YAML

exercises (RQ5–RQ6).

For deployment, retrieval scores alone separate answerable from out-of-scope queries with an AUROC of at least 0.907 across the four pipeline versions (RQ7), and preparation cost is paid only once at index time: local retrieval stays in the same low-millisecond band in every version, so the ranking gains of V3b come at a query latency indistinguishable from that of the naive baseline (RQ8).

Implications. These results support the Data-Centric AI claim that data preparation has a large effect on retrieval quality before a single model parameter is optimised. Once index size is isolated, the structural preparation stage improves retrieval on every metric, and generative keyword enrichment succeeds where statistical enrichment fails due to poor interaction with the specifics of this pipeline design. What that does *not* permit is attributing the improvement to the heading boundaries alone, since V2 changes several preparation decisions at once and the control separates that bundle only from chunk size (Section 6.5). Methodologically, the thesis shows that the chunk-level ranking metrics we defined are not granularity-robust: any preparation step that changes chunk size also changes the index size against which absolute ranks are measured, so such steps must be paired with a granularity-matched control before a metric drop is read as a retrieval failure. Without that control this study would have reported a real improvement as a regression. Practically, the recommendation for the Arcana ingest is the V3b configuration: structural chunking with hierarchy embedding, LLM-generated keyword headers and the Curated Tutoring Collection as a dynamically curated corpus extension, combined with a score-based rejection stage for out-of-scope questions. The recommendation is a qualified one: V3b is the strongest version on ranking and routing but not on coverage, where it does not improve on V2, so what the keyword headers buy is a better ordering of evidence V2 already retrieved (Section 6.4).

Future work. The evaluation stops at the retrieval stage, so whether the wider rank spread of the finer-chunked versions affects what the generator actually produces could not be observed. Closing the pipeline and measuring end-to-end whether the retrieval gains become more accurate and grounded tutoring answers is therefore the most direct next step.

The granularity-matched control could establish the direction of the corpus-size bias in the rank-based metrics but not its magnitude. A metric decaying with the rank percentile ($\text{rank}/N_{\text{chunks}}$) would quantify it exactly, requiring the pipeline to store the full ranked list per query instead of a top-k cutoff.

The multi-hop bottleneck that structural chunking had caused was only partially recovered by LLM keywords. What remains is not addressable by data preparation at all: it concerns how the retrieved list is processed before it reaches the generator, a stage the DCAI design keeps constant.

The out-of-scope separation is based on only eleven queries, which is enough to show that the signal exists but not to calibrate the rejection threshold a deployment would need, so a larger out-of-scope set is required before RQ7 can be answered reliably.

Finally, all measurements are done on a single German-language C programming corpus and one fixed model configuration. The *direction* of the findings (structure over granularity, generative over statistical enrichment, coverage over ranking) can be read as transferable, but only corpora from other courses and disciplines can show which effects are due to structural chunking and which to this corpus.

References

[1] Alibaba Group. *zVec: An In-Process Vector Database*. https://github.com/alibaba/zvec. Apache-2.0 licence. Accessed: 2026-03-29. 2026.

[2] Uday Allu, Biddwan Ahmed, and Vishesh Tripathi. *Beyond Extraction: Contextualising Tabular Data for Efficient Summarisation by Language Models*. 2024. arXiv: 2401.02333 [cs.CL]. URL: https://arxiv.org/abs/2401.02333.

[3] Mohammad Usman Altam, Md. Imtiaz Habib, and T. Hoang. *Optimizing Agricultural Research: A RAG-Based Approach to Mycorrhizal Fungi Information*. 2025. arXiv: 2511.14765 [cs.CL]. URL: https://arxiv.org/abs/2511.14765.

[4] Akari Asai et al. "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection". In: *The Twelfth International Conference on Learning Representations (ICLR)*. 2024. URL: https://openreview.net/forum?id=hSyW5go0v8.

[5] Christoph Auer et al. *Docling Technical Report*. 2024. arXiv: 2408.09869 [cs.CL]. URL: https://arxiv.org/abs/2408.09869.

[6] Jaime Carbonell and Jade Goldstein. "The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries". In: *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '98. Melbourne, Australia: Association for Computing Machinery, 1998, pp. 335–336. ISBN: 1581130155. DOI: 10.1145/290941.291025.

[7] Chi-Min Chan et al. *RQ-RAG: Learning to Refine Queries for Retrieval Augmented Generation*. 2024. arXiv: 2404.00610 [cs.CL]. URL: https://arxiv.org/abs/2404.00610.

[8] Mathieu Ciancone, Clovis Varangot-Reille, and Marion Schaeffer. *ChunkNorris: A High-Performance and Low-Energy Approach to PDF Parsing and Chunking*. 2025. arXiv: 2602.00010 [cs.IR]. URL: https://arxiv.org/abs/2602.00010.

[9] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods". In: *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '09. Association for Computing Machinery, 2009, pp. 758–759. DOI: 10.1145/1571941.1572114.

[10] Tom Fawcett. "An Introduction to ROC Analysis". In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010.

[11] Yunfan Gao et al. *Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks*. 2024. arXiv: 2407.21059 [cs.CL]. URL: https://arxiv.org/abs/2407.21059.

[12] Maarten Grootendorst. *KeyBERT: Minimal Keyword Extraction with BERT*. Version v0.3.0. 2020. DOI: 10.5281/zenodo.4461265.

[13] James A. Hanley and Barbara J. McNeil. "The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve". In: *Radiology* 143.1 (1982), pp. 29–36. DOI: 10.1148/radiology.143.1.7063747.

[14] Matthew Honnibal et al. *spaCy: Industrial-strength Natural Language Processing in Python*. 2020. DOI: 10.5281/zenodo.1212303.

[15] Anette Hulth. "Improved Automatic Keyword Extraction Given More Linguistic Knowledge". In: *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2003, pp. 216–223. DOI: 10.3115/1119355.1119383.

[16] Mragisha Jain et al. *Retrieval-Augmented Tutoring for Algorithm Tracing and Problem-Solving in AI Education*. 2026. arXiv: 2605.12988 [cs.AI]. URL: https://arxiv.org/abs/2605.12988.

[17] Johannes Jakubik et al. "Data-Centric Artificial Intelligence". In: *Business & Information Systems Engineering* 66.4 (2024), pp. 507–515. DOI: 10.1007/s12599-024-00857-8.

[18] Mohammad Hossein Jarrahi, Ali Memariani, and Shion Guha. "The Principles of Data-Centric AI". In: *Communications of the ACM* 66.8 (2023), pp. 84–92. DOI: 10.1145/3571724.

[19] Kalervo Järvelin and Jaana Kekäläinen. "IR Evaluation Methods for Retrieving Highly Relevant Documents". In: *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '00. Athens, Greece: Association for Computing Machinery, 2000, pp. 41–48. ISBN: 1581132263. DOI: 10.1145/345508.345545.

[20] Soyeong Jeong et al. "Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity". In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024, pp. 7036–7050. DOI: 10.18653/v1/2024.naacl-long.389.

[21] Nikhil Kandpal et al. "Large Language Models Struggle to Learn Long-Tail Knowledge". In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 2023. arXiv: 2211.08411.

[22] Vladimir Karpukhin et al. *Dense Passage Retrieval for Open-Domain Question Answering*. 2020. arXiv: 2004.04906 [cs.CL]. URL: https://arxiv.org/abs/2004.04906.

[23] Claas Kochanke. "Optimization and Performance Engineering of a Retrieval-Augmented Generation (RAG) System". Master's thesis. Georg-August-Universität Göttingen, 2026.

[24] Sushant Kumar et al. "Opportunities and Challenges in Data-Centric AI". In: *IEEE Access* 12 (2024), pp. 33173–33189. DOI: 10.1109/ACCESS.2024.3369417.

[25] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481Paper.pdf.

[26] Zhaoxing Li et al. *TutorLLM: Customizing Learning Recommendations with Knowledge Tracing and Retrieval-Augmented Generation*. 2025. arXiv: 2502.15709 [cs.IR]. URL: https://arxiv.org/abs/2502.15709.

[27] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[28] Varun Magesh et al. "Hallucination-Free? Assessing the Reliability of Leading AI Legal Research Tools". In: *Journal of Empirical Legal Studies* 22.2 (2025), pp. 216–242. DOI: 10.1111/jels.12413.

[29] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5.

[30] Agada Joseph Oche et al. *A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions*. 2025. arXiv: 2507.18910 [cs.CL]. URL: https://arxiv.org/abs/2507.18910.

[31] Arnau Pérez and Xavier Vizcaino. *Advanced Ingestion Process Powered by LLM Parsing for RAG System*. 2024. arXiv: 2412.15262 [cs.CL]. URL: https://arxiv.org/abs/2412.15262.

[32] Zackary Rackauckas, Arthur Câmara, and Jakub Zavrel. *Evaluating RAG-Fusion with RAGElo: An Automated Elo-Based Framework*. 2024. arXiv: 2406.14783 [cs.IR]. URL: https://arxiv.org/abs/2406.14783.

[33] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.

[34] Zarreen Reza et al. *Small Models, Big Support: A Local LLM Framework for Educator-Centric Content Creation and Assessment with RAG and CAG*. 2025. arXiv: 2506.05925 [cs.CY]. URL: https://arxiv.org/abs/2506.05925.

[35] Kunal Sawarkar, Abhilasha Mangal, and Shivam Raj Solanki. "Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers". In: *2024 IEEE 7th International Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, 2024, pp. 155–161. DOI: 10.1109/MIPR62202.2024.00031.

[36] Tim Schopf. *KeyphraseVectorizers: Set of Vectorizers that Extract Keyphrases with Part-of-Speech Patterns from a Collection of Text Documents*. https://github.com/TimSchopf/KeyphraseVectorizers. Accessed: 2026-04-01. 2022.

[37] Spurthi Setty et al. *Improving Retrieval for RAG Based Question Answering Models on Financial Documents*. 2024. arXiv: 2404.07221 [cs.CL]. URL: https://arxiv.org/abs/2404.07221.

[38] Ishneet Sukhvinder Singh et al. *ChunkRAG: Novel LLM-Chunk Filtering Method for RAG Systems*. 2025. arXiv: 2410.19572 [cs.CL]. URL: https://arxiv.org/abs/2410.19572.

[39] Nandan Thakur et al. "BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Vol. 1. 2021. arXiv: 2104.08663. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html.

[40] Yining Wang et al. "A Theoretical Analysis of NDCG Type Ranking Measures". In: *Proceedings of the 26th Annual Conference on Learning Theory*. Ed. by Shai Shalev-Shwartz and Ingo Steinwart. Vol. 30. Proceedings of Machine Learning Research. PMLR, 2013, pp. 25–54. URL: https://proceedings.mlr.press/v30/Wang13.html.

[41] Antonio Jimeno Yepes et al. *Financial Report Chunking for Effective Retrieval Augmented Generation*. 2024. arXiv: 2402.05131 [cs.CL]. URL: https://arxiv.org/abs/2402.05131.

[42] Daochen Zha et al. *Data-centric Artificial Intelligence: A Survey*. 2023. arXiv: 2303.10158 [cs.LG]. URL: https://arxiv.org/abs/2303.10158.

[43] Yanzhao Zhang et al. *Qwen3 Embedding: Advancing Text Embedding and Reranking through Foundation Models*. 2025. arXiv: 2506.05176 [cs.CL]. URL: https://arxiv.org/abs/2506.05176.

A Code Listings

This section reproduces three functions from the evaluation pipeline verbatim (only docstrings are shortened to a one-line comment). They are included because these three pieces of code define quantities the thesis relies on directly: the chunk boundaries that are the independent variable of the whole study (Listing 1), and the two score-weighted metrics introduced for this benchmark (Listings 2 and 3).

```python
def chunk_markdown_structural(md: str, doc_id: str) -> List[Block]:
    """V2/V3 chunking: heading-aware, atomic code blocks, 20% overlap."""
    import re

    MIN_TOKENS    = 500
    MAX_TOKENS    = 1500
    OVERLAP_RATIO = 0.20

    def approx_tokens(text: str) -> int:
        return len(text.split())

    lines = md.splitlines()
    raw_chunks: List[tuple] = []  # (section_path, text)
    section_stack = []
    buf = []
    in_code = False

    def flush(force: bool = False) -> None:
        text = "\n".join(buf).strip()
        if not text:
            buf.clear()
            return
        if force or approx_tokens(text) >= MIN_TOKENS:
            section_path = " > ".join(section_stack) if section_stack else None
            raw_chunks.append((section_path, text))
            buf.clear()

    for line in lines:
        if line.strip().startswith("```"):
            in_code = not in_code
            buf.append(line)
            continue

        if in_code:
            buf.append(line)
            continue

        m = re.match(r"^(#{1,6})\s+(.*)$", line.strip())
        if m:
            flush(force=True)
            level = len(m.group(1))
            title = m.group(2).strip()
            section_stack = section_stack[:level - 1] + [title]
            continue

        if not line.strip():
            if approx_tokens("\n".join(buf)) >= MIN_TOKENS:
                flush(force=True)
            else:
                buf.append("")
            continue

        buf.append(line)

        if approx_tokens("\n".join(buf)) >= MAX_TOKENS:
            flush(force=True)

    flush(force=True)

    blocks = []
    for i, (section_path, text) in enumerate(raw_chunks):
```

```
        if i > 0:
            prev_words = raw_chunks[i - 1][1].split()
            n = int(len(prev_words) * OVERLAP_RATIO)
            if n > 0:
                text = " ".join(prev_words[-n:]) + "\n\n" + text

        if i < len(raw_chunks) - 1:
            next_words = raw_chunks[i + 1][1].split()
            n = int(len(next_words) * OVERLAP_RATIO)
            if n > 0:
                text = text + "\n\n" + " ".join(next_words[:n])

        if section_path:
            text = f"# {section_path}\n\n{text}"

        blocks.append(Block(
            doc_id=doc_id,
            text=text,
            meta={"section_path": section_path} if section_path else {},
            section_path=section_path,
        ))

    return blocks
```

Listing 1: V2/V3 structural chunking (`Code/loaders.py`). Splits at heading boundaries, keeps code fences atomic, and applies a 20 % sliding overlap between neighbouring chunks. `MIN_TOKENS` only forces a flush *within* a section; a new heading (`force=True`) always starts a new chunk regardless of size, which is why the resulting chunk sizes vary so widely (Table 1).

```
def _rank_in_topk(evidence: str, top_k_indices: list[int],
                  corpus_texts: list[str]) -> int | None:
    """Return 1-based rank of first chunk containing evidence, or None."""
    for rank, idx in enumerate(top_k_indices, start=1):
        if _has_evidence(evidence, corpus_texts[idx]):
            return rank
    return None


def evaluate_chunk_level(
    gold_references: list[dict],
    top_k_indices: list[int],
    corpus_texts: list[str],
    rank_decay: str = "reciprocal",
) -> dict:
    """WRS at K = sum(matching_score_i * decay(rank_i)) / sum(matching_score_i)."""
    evidence_details    = []
    score_sum           = 0.0
    weighted_topk_score = 0.0

    for ref in gold_references:
        evidence       = ref.get("evidence", "").strip()
        matching_score = float(ref.get("matching_score", 1.0))

        if not evidence:
            continue

        score_sum += matching_score

        rank             = _rank_in_topk(evidence, top_k_indices, corpus_texts)
        evidence_indices = [i for i in top_k_indices
                            if _has_evidence(evidence, corpus_texts[i])]

        if rank is not None:
            decay = 1.0 / rank if rank_decay == "reciprocal" else 1.0 / math.log2(rank + 1)
            weighted_topk_score += matching_score * decay

        evidence_details.append({
            "evidence_snippet":       evidence[:80],
            "matching_score":         matching_score,
            "topk_has_evidence":      rank is not None,
```

```
            "topk_rank":              rank,
            "evidence_chunk_indices": evidence_indices,
        })

    wrs = (weighted_topk_score / score_sum) if score_sum > 0 else 0.0

    return {
        "wrs":              round(wrs, 4),
        "evidence_details": evidence_details,
    }
```

Listing 2: WRS@K computation for a single query (`Code/validation.py`). References without an evidence snippet are skipped entirely (line 15); references whose evidence is not found in the top-k contribute their `matching_score` to the denominator but nothing to the numerator (line 21), which is what makes WRS sensitive to how many references a query carries (Section 6.1, Appendix C). `_has_evidence` implements the exact-substring/LCS matching described in Section 4.6 and is omitted here.

```
def _chain_mrr(evidence_details: list, top_k_indices: list[int]) -> float | None:
    """1/chain_rank; 0.0 if a Ghigh reference is missing; None if no Ghigh exist."""
    ghigh = [ed for ed in evidence_details if ed["matching_score"] >= HIGH_SCORE_THRESHOLD]
    if not ghigh:
        return None

    ranks = []
    for ed in ghigh:
        idx_set = set(ed["evidence_chunk_indices"])
        rank = next(
            (pos for pos, chunk_idx in enumerate(top_k_indices, start=1) if chunk_idx in idx_set),
            None,
        )
        if rank is None:
            return 0.0  # missing reference -> Chain-MRR = 0 (applicable, failed)
        ranks.append(rank)

    return round(1.0 / max(ranks), 4)
```

Listing 3: Chain-MRR computation (`Code/reporting.py`). Returns `None` when a query has no G_{high} reference (not applicable, excluded from the mean), `0.0` when at least one G_{high} reference is missing from the top-k (applicable, failed), and otherwise the reciprocal of the *worst* (highest) rank among the G_{high} references – the bottleneck that motivates reporting Chain-MRR alongside Chunk-MRR (Section 4.6.1).

B Empirical Validation of WRS

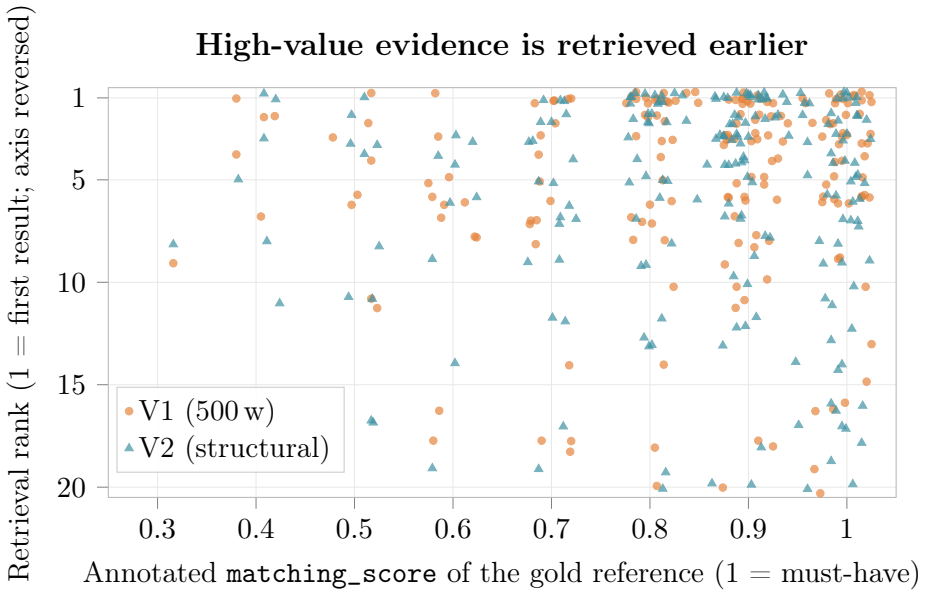

Figure 12: Importance vs. retrieval rank for every found gold reference (without CTC, V1 and V2). The y-axis is reversed so higher means retrieved earlier; points are jittered to separate the discrete score and rank values. High-importance references (right) concentrate at top ranks in both versions—$G_{\geq 0.8}$ evidence is found at median rank 3 (V1) / 4 (V2) versus 6 / 5 for lower-weighted references—confirming that the retriever prioritises must-have evidence; the slightly deeper V2 cloud reflects the evidence dispersion analysed in RQ2 (Section 5.2).

C WRS and Annotation Density

Section 6.1 argues that WRS is depressed for queries that carry many gold references, independently of retrieval quality, because the metric averages the rank decay over all references without normalising for how many there are. Table 4 reports the Pearson correlation between the reference count of a query and its WRS@20, together with the same correlation for Chunk-MRR, which scores only the first relevant chunk and therefore carries no such term. All values are computed on the 72 in-scope queries of the without-CTC condition.

Table 4: Correlation between the number of gold references per query and two chunk-level metrics ($n = 72$ in-scope queries, without CTC). WRS is consistently and negatively correlated with the reference count, Chunk-MRR is not.

Version	$r(n_{\text{refs}},$ WRS@20)	$r(n_{\text{refs}},$ Chunk-MRR)
V1 (500 w)	−0.137	+0.053
V2 (structural)	−0.186	+0.021
V3a (KeyBERT)	−0.207	+0.025
V3b (LLM)	−0.190	+0.017

The effect is visible directly in the group means. Among the complementary queries in V1, those with four to six references are retrieved better than those with two or three by Chunk-MRR (0.725 against 0.621) and worse by WRS (0.335 against 0.450); the four complementary queries with seven or more references fall to a WRS of 0.147 at a Chunk-MRR of 0.373. The two most densely annotated queries in the benchmark, both redundant and both carrying fourteen references, reach a mean WRS of 0.013 and 0.001 across the four versions.

Because the annotation is identical in every configuration, this sensitivity cannot produce a false difference *between* versions. A sensitivity check confirms this for the one comparison in which the densely annotated queries could plausibly have driven the result: the V1 → V2 WRS drop on the redundant subset (RQ5) is −0.048 over all 18 queries and −0.053 once the two fourteen-reference queries are excluded, so the drop is a dispersion effect rather than an artefact of annotation density.

D Keyword Quality Inspection

Table 5: Keyword sets injected in V3a and V3b, measured on the cached chunk files (451 chunks in each version). Counts are per extracted phrase, not per comma-separated token: four V3a phrases contain literal commas and would otherwise be counted more than once. "Markup or code characters" counts keywords containing at least one of `# { } ( ) < > | " ; %` or consisting solely of a C syntax token. The five most frequent V3a keywords are `# kapitel` (104 occurrences), `c-code` (87), `compiler` (46), `# include < stdio.h >` (41) and `# anhang` (32); for V3b they are `dynamischer Speicher` (108), `Heap` (63), `int` (60), `malloc` (59) and `printf` (57).

	V3a (KeyBERT)	V3b (LLM)
Keywords in total	3179	4095
Keywords per chunk (mean)	7.00	9.08
Distinct keywords	1639	1699
Mean keyword length (words)	1.56	1.24
Containing markup or code characters	18.8 %	1.2 %

Table 6 contrasts the keyword headers injected by V3a and V3b for three chunks, chosen to illustrate the three failure modes discussed in Section 6.3: re-extraction of the injected structural header, selection of conversion artefacts, and the loss of short but decisive tokens in favour of longer paraphrases. The keywords are reproduced verbatim from the cached chunk files.

Table 6: Injected keyword headers for three chunks, V3a (KeyBERT) against V3b (LLM), reproduced verbatim from the cached chunk files including the conversion artefacts that KeyBERT selects. Long literals are truncated with … where marked.

(a) `programmierung_mit_ansi_c-1-16.pdf` — table of contents	
V3a	`# programmierung`, `ansi-c`, `-operator`, `|---...---|` (a 260-character table rule), `makros`, `standardbibliothek`, `attribut const`
V3b	Stringvariablen, Stringkonstanten, Zeichen, Strings, sizeof-Operator, Castoperatoren, Typumwandlung, Nebeneffekte, Priorität der Operatoren, Kommaoperator
(b) `programmierung_mit_ansi_c-185-192.pdf` — A.5 Dateiverwaltung	
V3a	working directory, `./a.out ¿ fur`, programmen, pfadname, arbeitsverzeichnis, `fur benutzer2`, dateiverwaltung
V3b	`pwd`, `cd`, Arbeitsverzeichnis, Pfadname, home directory, Verzeichnisdatei, Dateibaum, Präfix
(c) `45.yaml` — solution chunk	
V3a	`printf("abcdefghijklmnopqrstuvwxyz[...` (unterminated string literal), `c-code`, `# include < stdio.h >`, `}`, `main`, `musterlösung`, `return`
V3b	`printf`, ASCII, Zeichenkette, Ausgabe, IO

Block (b) is the clearest case. Both extractors describe the same passage on the working

directory, but only the LLM emits the two tokens a student would actually type, `pwd` and `cd`; KeyBERT drops them below its minimum character length and returns the paraphrases *working directory* and *arbeitsverzeichnis* instead, which a query such as "was macht `cd`?" cannot match lexically. Block (a) shows the artefact mechanism, where a 260-character Markdown table rule is selected as a keyphrase because it is maximally distinctive across the corpus, and block (c) shows both on exercise material: V3a reproduces a full `printf` literal and three C syntax tokens, V3b names the concept the exercise practises.

E LLM Prompt for V3b

The following prompt was used for keyword generation in V3b. It was executed once per chunk; the model responded deterministically (temperature = 0) with a JSON list, which was stored in a cache file (`keywords_llm_v3b.json`) to keep V3b reproducible without repeated LLM calls. A SHA-256 hash of the prompt template is verified at startup to prevent cache contamination through unintentional prompt changes.

Prompt-Template `V3b`

```
_PROMPT_TEMPLATE = (
   "Du bist ein Experte für die Lehre der Programmiersprache C an Universitäten.
    Du erhältst einen Textabschnitt aus einem C-Programmierkurs für Studierende.
    Aufgabe: Generiere zwischen 5 und 10 fachliche Keywords, die den Inhalt
    charakterisieren. Diese Keywords werden zur Indexierung der Passage in einer
    Wissensdatenbank verwendet und verbessern die Auffindbarkeit der Passage durch
    ein Retrieval-System.
    Regeln:
    - Nur Begriffe, die im Text vorkommen oder direkt aus dem Inhalt ableitbar sind.
     Erfinde keine Konzepte hinzu.
    - Bevorzuge fachsprachliche C-Termini (malloc, struct, Pointer, Heap, ...)
    gegenüber Alltagssprache.
    - Mehrwort-Phrasen sind erlaubt ("Call by Reference"), maximal 3 Wörter.
    - Keine Duplikate, keine Synonyme derselben Idee.\n"
    - Antworte AUSSCHLIESSLICH mit einer JSON-Liste von Strings,
    ohne Erklärungstext, ohne Markdown-Codeblock-Marker.
    Beispiel einer gültigen Antwort:
    '["malloc", "dynamischer Speicher", "free", "Heap", "Speicherleck"]
    Textabschnitt {chunk_text}'
```

The prompt is written in German, as the corpus and benchmark queries are German, too. The JSON response was parsed via `json.loads`.

F Full Retrieval Metrics

For completeness, Table 7 and Table 8 report every metric computed by the evaluation pipeline for all five data-preparation versions (V1, the V1-215 control, V2, V3a, V3b) under both conditions (with and without the CTC).

Table 7: Full retrieval metrics by version and CTC condition. All @k metrics are evaluated at the cut-offs given in the header; all values are ratios in $[0, 1]$, Hit@k and Strict@20 being shares of queries. n = evaluated queries; the with-CTC rows average over all 83 queries and therefore include the eleven CTC-only queries that are out of scope in the without-CTC rows. Chain-MRR* is defined only on complementary queries ($n = 54$ with CTC, $n = 50$ without); Recall@20 and Strict@20 on the queries carrying at least one G_{high} reference ($n = 83$ with CTC, $n = 66$ without).

			Doc-level			Chunk-level						
Version	CTC	n	Hit@1	Hit@5	MRR	nDCG	WRS	WRec	Rec.	Strict	Chunk	Chain*
V1	yes	83	0.614	0.916	0.750	0.494	0.394	0.759	0.824	0.699	0.646	0.242
V1	no	72	0.569	0.903	0.706	0.412	0.303	0.706	0.787	0.621	0.573	0.204
V1-215	yes	83	0.590	0.916	0.732	0.459	0.357	0.688	0.734	0.554	0.600	0.188
V1-215	no	72	0.528	0.889	0.681	0.372	0.260	0.624	0.684	0.470	0.519	0.142
V2	yes	83	0.651	0.928	0.772	0.586	0.392	0.775	0.834	0.711	0.667	0.220
V2	no	72	0.597	0.889	0.719	0.526	0.287	0.752	0.816	0.667	0.585	0.182
V3a	yes	83	0.639	0.952	0.773	0.591	0.387	0.776	0.833	0.711	0.656	0.196
V3a	no	72	0.597	0.917	0.721	0.521	0.274	0.733	0.798	0.636	0.572	0.164
V3b	yes	83	0.687	0.928	0.799	0.611	0.405	0.786	0.841	0.723	0.696	0.238
V3b	no	72	0.681	0.861	0.765	0.546	0.300	0.749	0.812	0.667	0.617	0.218

Table 8: Out-of-scope separability (AUROC) and latency components by version and CTC condition. AUROC is defined only without the CTC, since with it all 83 queries are in scope (the 11 CTC-only queries become out-of-scope only when the collection is removed). "Infeas." counts queries whose gold evidence no chunk of the index contains. Latencies in milliseconds.

					Embedding		End-to-end	
Version	CTC	n	Infeas.	AUROC	mean	P95	mean	P95
V1	yes	83	4	—	3699	32365	3712	32380
V1	no	72	4	0.907	3273	45292	3290	45324
V1-215	yes	83	17	—	3886	49292	3896	49296
V1-215	no	72	17	0.948	3169	40273	3175	40283
V2	yes	83	0	—	3167	32322	3186	32359
V2	no	72	0	0.924	4297	36344	4317	36371
V3a	yes	83	0	—	2921	20196	2940	20220
V3a	no	72	0	0.909	3781	42297	3800	42317
V3b	yes	83	0	—	2967	6712	2980	6742
V3b	no	72	0	0.928	3295	42309	3309	42322

Local retrieval latency, the only component that data preparation could plausibly affect, is reported in Section 5.8; it stays between 14 and 20 ms in every configuration.