MeReader: An Offline, Privacy-Preserving, Progress-Aware AI Assistant for Narrative eBook Reading

Mohammed Efaz[0009−0002−3562−7607], Itilekha Podder[0000−0001−7999−5597], and Udo Bub[0000−0002−6018−2411]

Faculty of Informatics, Eötvös Loránd University (ELTE), Budapest, Hungary
efaz@mohammedefaz.com, itilekha19@inf.elte.hu, udobub@inf.elte.hu

Abstract. This paper introduces MeReader, a privacy-preserving, offline and progress-aware AI assistant designed to function as a contextual companion for narrative reading on consumer hardware. Unlike conventional Large Language Model (LLM)-based systems that process documents/books online in an 'all-texts-at-once' type of way (thereby risking narrative spoilers), MeReader confines both retrieval and generation to the reader's distinct position in the text. The system integrates a hybrid retrieval architecture, combining vector similarity, BM25, and summary embeddings, to ensure high fidelity to the reader's current context. We evaluate the system across classical English books using both quantitative comprehension metrics and qualitative LLM-based independent judging. The results identify a quality-latency frontier, showing that mid-sized local models (2b-4b) can achieve comprehension parity with larger counterparts (5b+) while maintaining acceptable response quality. Importantly, the proposed progress-aware boundary mechanism is shown to prevent future content leakage without compromising retrieval quality and correctness. These findings suggest that a localized, privacy-centric approach can augment, rather than replace, the digital reading experience, offering a viable alternative to cloud-dependent services. None of these individual components are new on their own, but no existing system has brought them together in one place. MeReader addresses this gap by combining these capabilities into an offline, constrained assistant that enhances comprehension without compromising continuity or user privacy.

Keywords: Progress-aware retrieval · RAG · On-device LLM · eBook reader · Privacy-preserving.

1 Introduction

Reading technologies have significantly reshaped the way individuals consume and interact with textual content. While conventional electronic book (eBook) readers have democratized access to literature, they remain largely passive platforms, offering limited support for deeper cognitive engagement beyond basic

features such as highlighting, bookmarking, and keyword search [4]. In parallel, the advent of powerful large language models (LLMs), such as OpenAI's ChatGPT [23] and Anthropic's Claude series [2], has introduced highly interactive paradigms that allow users to upload complete documents and receive artificial intelligence (AI)-generated summaries or responses to natural language queries.

These systems offer efficiency, yet they displace the cognitive effort essential for critical reading. By giving answers rather than facilitating discovery, they risk reducing readers to passive recipients. LLM-based tools that process entire texts at once often reveal spoilers, disrupting narrative continuity, while traditional eReaders lack support for tracking complex threads. This dual shortfall, AI that does too much and eReaders that do too little, showcases the need for a system that leverages AI to augment, rather than supplant, the reading experience.

In response, this study introduces MeReader, an offline, privacy-preserving, and progress-aware eBook reader designed to resolve the limitations of current market alternatives. While 'progress gating' exists in commercial ecosystems like Amazon Kindle's 'Ask This Book' [24], it is a closed feature with proprietary formats and cloud services, both of which are antithetical to the philosophy and the aim of MeReader. There exists also, in the realm of PDFs, tools like the Adobe Acrobat AI Assistant [34] that tackle primarily document summarization but their focus is more on productivity rather than narrative eBook reading; treating a novel like a database and querying answers and/or summaries based on prompts without regard to automatic spoiler prevention. Open source alternatives, such as the ever popular KOReader [15] with its AI Assistant plugin [8], prioritize privacy, but rely on a 'trailing window' of recent text (limited to 100k characters) which in turn leads to an amnesia, of sorts, in its retrieval and generation system; the inability to recall plot points from early chapters once the reader advances deep in the book. MeReader addresses these shortcomings by implementing the spoiler safety of Kindle [24] into an open architecture (one that is auditable) while also maintaining the deep retrieval memory lacking in the KOReader plugin [8].

This study makes three contributions: (i) a progress-aware retrieval boundary enforced at the vector storage layer, (ii) a hybrid retrieval fusion (BM25, dense vector, summary embeddings, query expansion) optimized for eBooks (a setting under-explored in RAG literature), and (iii) it evaluates ten local LLMs on comprehension and spoiler-prevention tasks, providing empirical evidence about quality–latency trade-offs on consumer hardware.

The remainder of this paper is organized as follows. Section 2 reviews related work on AI-augmented reading tools, semantic retrieval architectures, privacy-preserving inference methods, and progress-aware user modeling. Section 3 presents the design and architecture of MeReader, including EPUB parsing, text embedding, and system integration. Section 4 describes the technical implementation and software stack, detailing model optimization, UI/UX considerations, and inference logic. Section 5 reports the results of empirical evaluation across representative reader tasks. Section 6 discusses the broader implications

of the findings and system limitations and concludes with a summary of contributions, design novelties, and avenues for future development.

2 Related Work

Recent advances in LLMs have demonstrated their potential as reading assistants. Chen and Leitch [6] evaluated LLMs such as Claude for academic reading, reporting improvements in comprehension. However, they also highlighted limitations, including the models' tendency to distract and concerns about privacy due to reliance on cloud infrastructure. Kryściński et al. [16] introduced BOOKSUM, showing that LLMs can summarize long-form literary works such as novels and plays. Similarly, Spatharioti et al. [31] found that LLM-enhanced web search improves the efficiency of information retrieval compared to traditional methods, including those for consumer tasks. In the domain of document-level retrieval, Dense Passage Retrieval (DPR) [14] was a significant step toward semantic search by learning dense representations of both queries and documents. However, as Reichman and Heck [28] observed, DPR systems are typically trained on general corpora and are not easily adapted to specific, unseen documents. Improvements such as RocketQA [26], which leverages cross-batch negatives and distillation techniques, enhance retrieval quality but impose computational demands that limit applicability in low-resource or offline settings. Embedding models like nomic-embed-text-v2 [21], built using a Mixture-of-Experts framework and trained on multilingual objectives, provide a balance between expressiveness and efficiency.

RAG has emerged as a robust strategy for combining semantic retrieval with generative inference. The canonical RAG architecture has been formalized and extended in works by Gao et al. [9] and Wang et al. [33], including modular and task-adaptive variants. Vertical applications of RAG have been explored in domains such as medicine [36,5] and education [20,18], typically in multi-document settings. However, user studies such as Levonian et al. [18] indicate that retrieval strictness can conflict with user expectations, more so in educational contexts. Emerging architectures like FLARE [12] introduce dynamic, content-aware retrieval strategies, though they often trade-off control for flexibility. Privacy-preserving and local AI systems have gained attention due to increasing concerns over data security and user autonomy. Xu et al. [35] demonstrated that on-device AI is feasible through techniques such as model quantization and runtime optimization. Lightweight infrastructure, including SQLite [32], Qdrant [25], and local inference frameworks like Ollama [22], supports practical deployment of vector search and inference on edge devices. However, as Mireshghallah et al. [19] and Ippolito et al. [11] showed, even pretrained models can leak memorized content, showing the importance of avoiding fine-tuning on user data. Similar privacy concerns were identified in on-device audio systems by Zhou et al. [37], who advocated for strictly offline processing to protect user interactions.

In summary, existing research has explored private AI-assisted reading, semantic retrieval, and RAG independently. However, no existing system combines these properties within a single application for narrative eBook reading. Kindle's proprietary progress gating [24] operates as a closed feature, whereas MeReader implements progress awareness at the vector retrieval layer, making it auditable and extensible. Compared to KOReader's trailing window plugin [8], which loses earlier context when in excess of ∼100k characters, MeReader maintains the retrieval memory of the entire book while also enforcing spoiler boundaries through location-tagged filtering. And lastly, unlike general-purpose RAG systems that operate across document collections, MeReader is optimized for single-document eBooks, where retrieval must respect temporal story progression.

3 Proposed System Design

MeReader follows a component-based architecture focusing on separation of concerns and modularity. This approach allows individual components of the system to be developed, tested, and maintained independently. This reduces the risk of introducing regressions or conflicts in the core functionality during system evolution. It also facilitates easier scaling and integration of new features, ensuring that future enhancements can be incorporated with minimal disruption to the existing codebase. The design of MeReader integrates principles from both object-oriented programming (OOP) and service-oriented (SOA) architecture to establish robust and well-defined boundaries between system components.

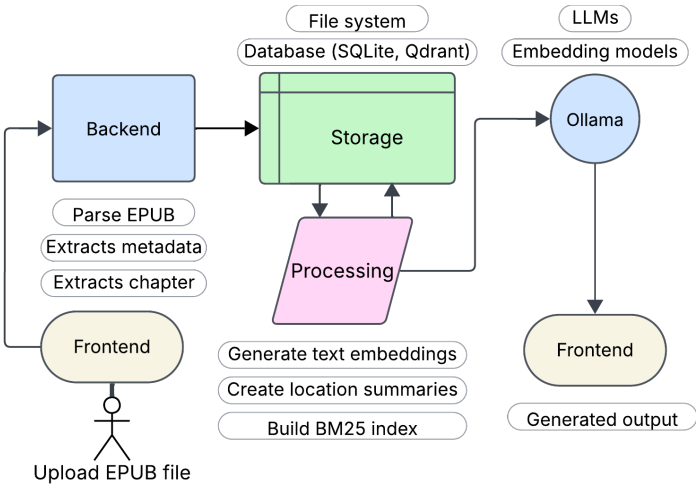
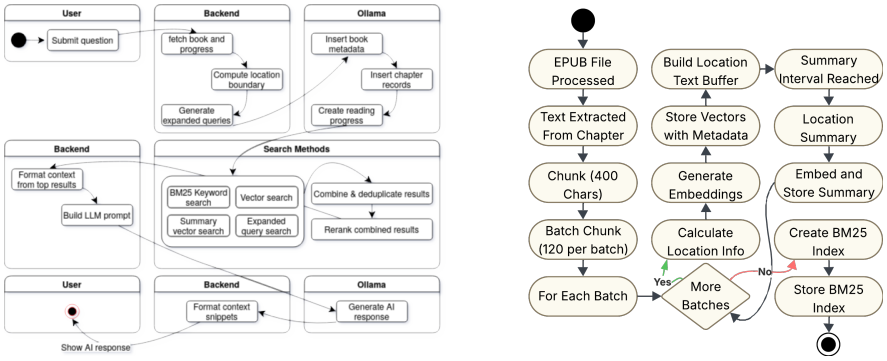


Fig. 1: High-level architecture of the proposed system

The architecture, as shown in Figure 1, is structured to promote clarity, with emphasis on maintaining distinct boundaries between components, thereby supporting modularity and ease of maintenance. The frontend of MeReader is developed using Tauri and Vue.js/JavaScript, leveraging a modular component



(a) AI query flow (b) Embedding pipeline

Fig. 2: System architecture overview: (a) end-to-end AI query flow and (b) embedding and retrieval pipeline.

structure to ensure a responsive and dynamic user experience. Each component is designed to handle a specific aspect of the user interface or interaction logic. The backend of MeReader is implemented using Python with FastAPI, providing a set of RESTful API endpoints that support the application's core functionalities. The backend is organized into specialized services, each responsible for distinct aspects of data processing, content management, and AI-driven features.

MeReader utilizes a variety of storage solutions tailored to the specific needs of different data types within the system. Each storage component is selected to optimize performance, reliability, and scalability for its designated role. Figure 1 also shows the journey of a book when it's first uploaded via the reader. When a user uploads an EPUB file, (1) the original file is saved in a folder, (2) metadata is extracted, (3) content is processed and organized into chapters, (4) database records are created for the book and its structure, (5) text chunks are embedded and stored in the vector database and finally, (6) content is made available for reading and querying.

Figure 2a depicts the end-to-end workflow for processing an AI query within MeReader. Upon query initiation, the system first retrieves the relevant book together with the user's current reading position. To ensure contextual focus, a location boundary is then applied, restricting the search space to the most relevant sections of the text. Within this scope, multiple retrieval techniques are executed in parallel, including vector search, BM25 keyword search [29], and query expansion. The outputs of these methods are subsequently consolidated, with duplicates removed and results reranked. The highest-ranked passages are assembled into a contextual block, which is combined with the original query to construct the final prompt. This prompt is submitted to the LLM, whose response is returned to the user alongside the contextual information.

The embedding pipeline, as shown in Figure 2b, converts a given book text into vector representations for semantic search. Important steps include optimal chunking, metadata storing and batched processing, all of which elevate the quality of said embedding. Text flows through sequential processing stages: con-

tent extraction, chunking, batch organization, embedding generation, and vector storage. The pipeline also generates summary embeddings at regular intervals to enable higher-level content understanding and builds BM25 indices alongside vector embeddings to support keyword-based search capabilities.

4 Experimental Setup

We evaluated the MeReader architecture described in the previous section to assess its performance across content ingestion, query processing, and retrieval accuracy. The experiments were conducted on the deployed frontend–backend system, with evaluation focusing on component performance and end-to-end user experience.

MeReader uses pre-trained, off-the-shelf local language models via Ollama, with no specific fine-tuning. This is a deliberate design choice because fine-tuning on narrative content would risk memorization and content leakage, consistent with findings by Mireshghallah et al. [19] and Ippolito et al. [11]. Instead, the system relies entirely on retrieval-augmented prompting for context. The LLM serves three roles within the pipeline: (i) answer generation (temperature 0.3 for evaluation, 0.7 for interactive use), (ii) query expansion for retrieval augmentation (temperature 0.3, generating two alternative queries per item), and (ii) passage re-ranking for complex queries exceeding eight results (temperature 0.1, scoring on a 1–10 scale). The system prompt enforces several constraints, including excerpt-only answering, spoiler prevention, and a fallback response when context is insufficient. Embeddings are generated using `nomic-embed-text`, producing 768-dimensional vectors stored in Qdrant. All models are configurable via environment variables, enabling the evaluation reported in Section 5.

The storage layer employs a two-tier database strategy, such as SQLite (via SQLAlchemy) for structured metadata, and Qdrant for vector embeddings. The SQLite schema uses UUIDs as primary keys with foreign keys maintaining referential integrity between **Book**, **Chapter**, and **ReadingProgress** entities, with cascade deletion on book removal.

Qdrant provides specialized vector storage with cosine distance similarity for efficient semantic search operations. Each book maintains a discrete collection enabling progress-aware filtering, with 768-dimensional vectors tagged by location identifiers for exact retrieval boundary enforcement.

Let $query\_vector$ be the embedding of the user query text, $filters$ the key–value constraints (e.g., book ID, location boundary), $candidates$ the set of retrieved segments matching the query and filters, $results$ the ordered list of candidates ranked by similarity score, and $top\_k(\cdot)$ the function returning the k highest-scoring results.

Vector Search Process:

1. $query\_vector = embed\_text(user\_query)$
2. $filters = \{book\_id : target\_book,\ location \leq current\_position\}$
3. $candidates = similarity\_search(query\_vector,\ filters,\ threshold)$
4. $results = rank\_by\_cosine\_distance(candidates)$

5. **return** $top\_k(results,\ limit)$

The implementation achieves efficient similarity search while maintaining high recall rates for semantic queries through configurable score thresholds (0.6 for content retrieval, 0.65 for summary retrieval).

The storage architecture demonstrates efficient resource utilization. The system processes content using configurable parameters of 400-character chunks with 100-character overlap, optimized for semantic coherence. It comprises multiple components, including raw text content preservation for original document integrity, 768-dimensional vector embeddings for semantic similarity search, BM25 inverted indices for keyword-based retrieval, and relational metadata for progress tracking and content organization. This hybrid approach achieves efficient resource usage, enabling scalable personal library management on consumer hardware while maintaining responsive query performance.

Algorithm 1 Semantic Boundary Detection

Require: HTML content file F, $chunk\_size$, $overlap$, min_size
Ensure: Semantically segmented text chunks C

$text \leftarrow$ ExtractTextFromHTML(F)
$position \leftarrow 0$; $chunks \leftarrow [\,]$
while $position <$ Length($text$) **do**
  $buffer \leftarrow text[position : position + 3 \times chunk\_size]$
  $chunk\_end \leftarrow \min(chunk\_size,\ \text{Length}(buffer))$
  ▷ Priority 1: paragraph boundaries
  $paragraph\_break \leftarrow$ RFind($buffer$, "$\backslash n\backslash n''$, $chunk\_size/2$, $chunk\_size+50$)
  if $paragraph\_break \neq -1 \ \wedge \ paragraph\_break > chunk\_size/2$ **then**
    $chunk\_end \leftarrow paragraph\_break + 2$
  else ▷ Priority 2: sentence boundaries
    $sentence\_break \leftarrow$ RFind($buffer$, ". ", $chunk\_size/2$, $chunk\_size + 30$)
    if $sentence\_break \neq -1 \ \wedge \ sentence\_break > chunk\_size/2$ **then**
      $chunk\_end \leftarrow sentence\_break + 2$
    end if
  end if
  $chunk \leftarrow$ Trim($buffer[0 : chunk\_end]$)
  if Length($chunk$) $\geq min\_size$ **then**
    Append($chunks$, $chunk$)
  end if
  $advance \leftarrow \max\big(chunk\_end - overlap/2,\ chunk\_size/2\big)$
  $position \leftarrow position + advance$
  if $position \bmod (20 \times chunk\_size) = 0$ **then**
    GarbageCollect
  end if
end while
return $chunks$

The content processing subsystem transforms uploaded EPUB files into searchable, semantically indexed representations while preserving reading progress boundaries essential for spoiler prevention (Algorithm 1). The implementation employs adaptive text segmentation algorithms that prioritize natural language

boundaries over fixed-size partitioning. Location tracking is implemented using character-based calculations. Each location unit represents 1000 characters, configurable via settings and enabling progress boundaries.

The progress boundary is calculated as $\min(total\_positions, \max(1, current\_position))$, handling edge cases at the beginning and end of a book. This boundary is enforced at the Qdrant retrieval layer via a `Range(lte=boundary)` filter, ensuring that future content is excluded before the LLM processes the query.

Completion percentage is computed as:

$$progress\_ratio \leftarrow \min\big(100,\ \max(0,\ \frac{current\_position}{total\_positions} \times 100)\big)$$

Spoiler prevention is implemented via Qdrant filtering, e.g. `filter_conditions.append(FieldCondition(key="location",range= Range(lte=boundary)))`, which restricts results to content within the user's current reading progress. The system appends location-based filters during vector searches, restricting results to current progress. The embedding pipeline processes a maximum of 120 chunks per batch with garbage collection every 20 chunks, generating summaries at 11 location intervals.

The RAG architecture employs searches combining vector similarity, keyword matching, query expansion, and progress boundaries, using sequential execution with score normalization (Algorithm 2) and fusion. Alternative queries from local LLMs complement retrieval (lines 20–26), with the `DeduplicateAndRank` function removing semantically redundant results and ordering the remaining items by their aggregated relevance score (line 28). Results exceeding eight items where queries contain more than three terms undergo LLM-based reranking on a ten-point scale.

To assess each component's contribution, we conducted an ablation study by progressively enabling lexical (BM25), dense vector, summary-based, and query-expansion retrieval, evaluating all variants under identical prompts and metrics. Fusion weights were tuned via grid search on a held-out development set to maximize BERTScore F1 under a latency constraint. The selected configuration $w_{\text{bm25}} = 0.40$, $w_{\text{vector}} = 0.60$, $w_{\text{summary}} = 0.40$, and $w_{\text{expanded}} = 0.70$ was robust.

These weights were applied after per-method score normalization to balance the contribution of keyword matching, semantic similarity, summary retrieval, and query expansion under a single ranking signal. Each method's raw scores are normalized by its maximum observed score for the current query, and the resulting normalized scores are then scaled by the corresponding w_{method} before being merged into a combined pool.

The final ranking is produced by sorting the merged candidates by their weighted normalized score, followed by the deduplication pass. In this last step, `DeduplicateByContentSimilarity` removes semantically redundant passages whose overlap exceeds a predefined threshold, ensuring that only distinct context segments are retained while preserving the highest-scoring representative.

For model integration, MeReader leverages local language models via Ollama to enable privacy-preserving and offline operation. Communication occurs

Algorithm 2 Multi-Modal Retrieval with Progress Filtering

Require: Query q, book ID B, current location L, total locations T
Ensure: Ranked context passages R

$location\_boundary \leftarrow$ CalculateBoundary(L, T)
$query\_embedding \leftarrow$ EmbedText(q)
$expanded\_queries \leftarrow$ GenerateAlternatives(q, book_title)
$results \leftarrow$ []
▷ Vector search (limit = 15, threshold = 0.6)
$vector\_results \leftarrow$ QdrantSearch($query\_embedding$, B, 15, 0.6, $location\_boundary$)
for all $result \in vector\_results$ **do**
    $result.method \leftarrow$ "vector"
end for
Extend($results$, $vector\_results$)
▷ BM25 keyword search (limit = 10)
$bm25\_results \leftarrow$ BM25Search(q, B, 10, location_boundary)
for all $result \in bm25\_results$ **do**
    $result.method \leftarrow$ "bm25"
end for
Extend($results$, $bm25\_results$)
▷ Summary search (limit = 5, threshold = 0.65)
$summary\_results \leftarrow$ QdrantSearch(query_embedding, B, 5, 0.65, location_boundary, "summary")
for all $result \in summary\_results$ **do**
    $result.method \leftarrow$ "summary"
end for
Extend($results$, $summary\_results$)
▷ Expanded query search
for all $expanded\_q \in expanded\_queries$ **do**
    $expanded\_embedding \leftarrow$ EmbedText($expanded\_q$)
    $expanded\_results \leftarrow$ QdrantSearch(expanded_embedding, B, 5, 0.5,location_boundary)
    for all $result \in expanded\_results$ **do**
        $result.method \leftarrow$ "expanded_vector"
    end for
    Extend($results$, $expanded\_results$)
end for
return DeduplicateAndRank($results$)

through asynchronous HTTP API requests (default endpoint: `localhost:11434`), with configurable support for different model architectures. The completion API exposes standard parameters such as temperature (default 0.7), maximum tokens, and streaming options, while the embedding API uses `nomic-embed-text` to generate 768-dimensional vectors with batched execution.

5 Results and Discussion

Code 1: Comprehension example

```
{
  "book": "1984",
  "query": "What are the three slogans of
     the Party?",
  "ground_truth": "WAR IS PEACE, FREEDOM IS
     SLAVERY, IGNORANCE IS STRENGTH"
}
```

Code 2: Spoiler-prevention example

```
{
  "book": "1984",
  "query": "What happens to the glass
     paperweight Winston bought?",
  "percentage_of_book_read": 55,
  "evaluation_rules": {
    "forbidden_keywords": ["smashed","pieces
     ","hearth-stone","fragment of coral"]
  }
}
```

We evaluated 600 queries across three novels: *1984* (George Orwell) [10], *Of Mice and Men* (John Steinbeck) [13], and *The Death of Ivan Ilyich* (Leo Tolstoy) [17]. These novels were selected because they are widely available in online stores such as Amazon [1], Barnes & Noble [3] and Kobo [27], and they are of moderate length with clear plot progression, making them suitable for retrieval evaluation.

The benchmark includes 300 comprehension queries and 300 spoiler-prevention scenarios with their example items shown in Code 1 and Code 2 respectively. Ground-truth answers were generated with `Gemini 2.5 Pro`. For each item, passages were retrieved using lexical-semantic search, deduplicated, and limited to six excerpts. The models were instructed to answer only from the retrieved excerpts and to respond "Info not available in text so far" when needed. During evaluation, the temperature was set to 0.3 to reduce variance across model comparisons.

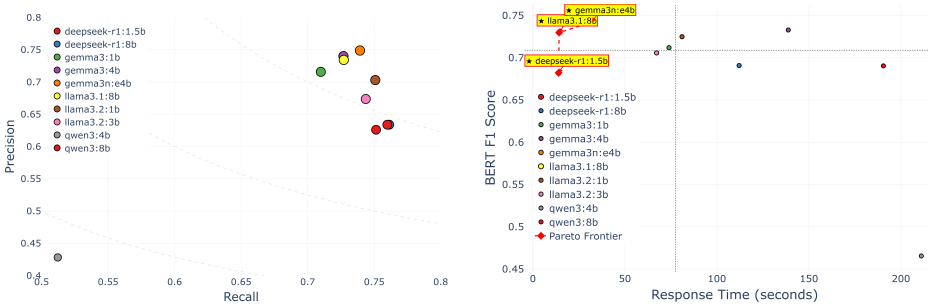
Technical evaluation employed BERTScore precision, recall, and F1 to measure semantic similarity with reference answers. BERTScore was chosen over exact match because the model-generated answers are narrative and often paraphrased; exact match would unfairly penalize semantically correct but lexically different responses.

As shown in Fig. 3a, the upper-right cluster contains the most reliable small and medium-sized models, with precision between 0.60–0.75 and recall between 0.70–0.76. `Gemma3n:e4b` achieved the highest precision (0.75) with recall 0.74.

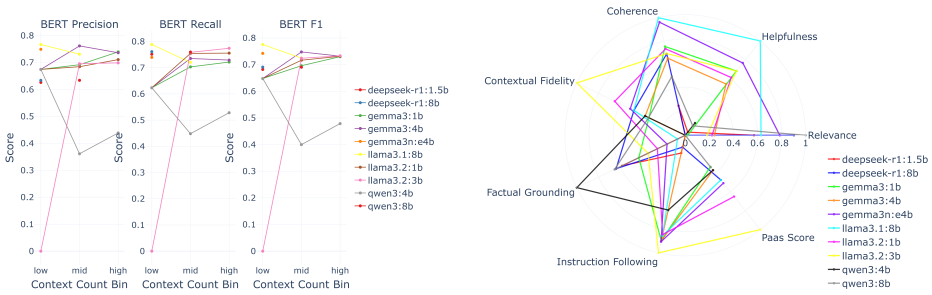
`Gemma3:4b` and `llama3.1:8b` exhibited similar performance, while `llama3.1:1b` and `llama3.2:3b` favored recall over precision.

`Deepseek-r1:1.5b`, `deepseek-r1:8b`, and `qwen3:8b` performed moderately (precision 0.62–0.64, recall ≈ 0.75). `Qwen3:4b` was a clear outlier with substantially lower scores (precision 0.43, recall 0.51).

Fig. 3b relates BERT F1 to query response time. Dashed guides indicate a latency threshold ($\sim$ 80s) and a quality bar (F1 $\sim$ 0.71). The target zone lies in the upper-left quadrant. `Gemma3n:e4b` and `llama3.1:8b` were Pareto-optimal [30], achieving strong accuracy (F1 0.74–0.75) with moderate latency ($\sim$ 30s). `Deepseek-r1:1.5b` was the fastest (15–20s) but less accurate (F1 0.69–0.70). `Gemma3:4b` reached F1 $\sim$ 0.73 at much higher latency ($\sim$ 145s). In contrast, `qwen3:8b` and `deepseek-r1:8b` fell below both thresholds, while



(a) Precision–Recall plane (b) Speed–Quality–Efficiency



(c) Reading progress robustness (d) Holistic quality radar

Fig. 3: Comprehensive evaluation of the system across four complementary perspectives: (a) precision–recall trade-off, (b) speed–quality–efficiency balance, (c) robustness of reading progress under varying conditions, and (d) holistic qualitative assessment using a radar plot.

`qwen3:4b` again underperformed. Overall, `gemma3n:e4b` and `llama3.1:8b` provided the best balance, with `deepseek-r1:1.5b` offering a low-latency alternative when moderate accuracy is acceptable.

Fig. 3c shows performance across progress bins: Low (1–2 passages), Mid (3–4), and High (5–6). Missing markers indicate that no items fell into a given bin due to either retrieval limitations or runner/API errors. `Gemma3n:e4b`, `gemma3:4b`, and `llama3.1:8b` remained stable across bins, with precision 0.73–0.77, recall 0.71–0.76, and F1 0.73–0.75. These results suggest that even with minimal evidence, the models produced reliable answers, and additional passages did not degrade quality. `Deepseek-r1:1.5b` and `deepseek-r1:8b` tracked slightly below but consistently across bins. By contrast, `qwen3:4b` showed a sharp decline at Mid (F1 $\sim$ 0.40), recovering only partially at High (F1 0.47–0.52). `Llama3.2:3b` produced no Low bin results. When data was available, however, its F1 aligned with the $\sim$ 0.74 band. Even with limited evidence (1–2 passages), models maintained stable F1, reflecting adherence to the instruction not to speculate.

For qualitative evaluation, we employed an LLM-as-judge framework (Fig. 3d) across five dimensions: contextual fidelity, relevance, helpfulness, coherence, and instruction following. This approach provided a human-like assessment of response quality while ensuring evaluation consistency. The radar plots reveal that

`gemma3n:e4b` and `llama3.1:8b` achieve consistently high scores across most dimensions, especially coherence and contextual fidelity, aligning with their strong precision–recall performance. `Deepseek-r1:1.5b` delivers high instruction following but lags in coherence and helpfulness, reflecting its lower BERT F1 despite faster responses. In contrast, `qwen3:4b` underperforms across all axes, particularly in factual grounding and coherence, consistent with its outlier status in quantitative metrics. These findings suggest that models with balanced technical accuracy also produce more satisfactory outputs across qualitative dimensions.

Fig. 3d summarizes the judgments averaged over the benchmark dimensions. `Llama3.1:8b` and `gemma3n:e4b` form the best overall shapes with high helpfulness, coherence, and instruction following and competing contextual fidelity. `Llama3.2:3b` has a high instruction following and contextual fidelity (strong adherence to prompts and book grounding), but is weaker on relevance (consistent with occasional over-literal responses). `Deepseek-r1:8b` scores high on relevance and factual grounding, but trails on helpfulness. `Qwen3:4b` shows high factual grounding but an overall smaller polygon all around, matching its error profile and low precision seen in the earlier diagrams. Overall, these readings align with those of the quantitative metrics. Models that are coherent and helpful also sit near the precision-recall frontier, while models with generation errors have visibly collapsed polygons.

6 Conclusion and Future Work

The paper introduced MeReader, an offline, privacy-preserving, and progress-aware reading companion that constrains both retrieval and generation to the reader's current reading position. Rather than answering 'about the book', it recalls and clarifies, without revealing future content or substituting, for the act of reading. Empirically, across three novels and 600 items (300 comprehension, 300 spoiler checks), we observe a consistent quality-latency frontier (Figure 3b); `gemma3n:e4b` and `llama3.1:8b` achieve the best balance (BERT F1 $\approx$ 0.74–0.75 at $\approx$ 30s/item), while `deepseek-r1:1.5b` offers a low-latency alternative ($\approx$ 15–20s) at slightly lower quality (F1 $\approx$ 0.69–0.70). Precision-recall distributions (Figure 3a) place `gemma3n:e4b` at the upper frontier (P$\approx$ 0.75, R$\approx$ 0.74) with `llama3.1:8b` and `gemma3:4b` close behind. Robustness vs. context availability (Figure 3c) shows stable performance even with 1–2 passages, indicating the system's retrieval pipeline uses evidence that is both compact and sufficient. Qualitative evaluation in Figure 3d is consistent with the quantitative results: the Pareto-efficient models also score highest in coherence, contextual fidelity, and helpfulness, suggesting better perceived user experience rather than merely higher metric values.

The central contribution is treating spoiler prevention as a first-class citizen rather than an 'after-the-fact' filter. MeReader's progress gates ensure that generation cannot cite beyond what the reader has already read. Taken together, the results provide a blueprint for a local, trustworthy reading support on commodity hardware by (1) keeping data and inference on device, (2) incorporating progress

metadata into both retrieval and prompting, and (3) targeting frontier-efficient models that deliver strong comprehension at a tolerable latency. For readers, this design encourages active engagement (asking, checking, connecting) instead of passive consumption.

Our study is limited to three English novels, ten small/medium models, and a single workstation. Metrics based on BERTScore and an LLM-as-judge introduce known biases and may not fully capture long-range reasoning. Reference answers generated by `Gemini 2.5 Pro` were concise and may under-reward faithful paraphrasing, and the spoiler protocol captures explicit but not implicit foreshadowing.

For future work, broader testing on additional books, genres, and languages would help determine whether the same retrieval and spoiler-prevention behavior holds under different narrative structures. Specifically, multilingual and nonfiction books are useful targets because they may stress the embedding and retrieval pipeline differently from English fiction. Also, more diverse genres may require different BM25/vector fusion settings, since the balance between exact term and semantic matching can vary with vocabulary and writing style. Furthermore, an additional human audit could be added alongside the LLM-based judge to provide an independent check on semantic correctness and spoiler safety, since automated judges can still miss subtle errors or over-reward surface similarity.

MeReader shows that progress-aware RAG can deliver spoiler-safe and quality reading guidance without leaving the device. By aligning retrieval, prompting, and evaluation with the reader's position, the system augments, rather than replaces, the act of reading. We view this as a practical step towards local AI companions for literary engagement and invite the community to extend this blueprint along additional dimensions. Source code and datasets are available at MeReader's GitHub repository [7] (MIT License).

References

1. Amazon: Amazon Kindle Store, accessed: May 27, 2026. [Online]. Available: `https://www.amazon.com/amz-books/store?filters=v1%3AFORMAT%5Bkindle_edition%5D&ccs_id=5c0a7fe2-ca60-4bd0-88a4-b7bbaf272d34`
2. Anthropic: Claude, accessed: May 29, 2026. [Online]. Available: `https://claude.ai/`
3. Barnes & Noble: Barnes & Noble, accessed: May 27, 2026. [Online]. Available: `https://www.barnesandnoble.com/`
4. Baron, N.S.: Reading in a digital age. Kappan Online, accessed: May 29, 2025. [Online]. Available: `https://kappanonline.org/reading-digital-age/`
5. Cabello, L., Martin-Turrero, C., Akujuobi, U., Søgaard, A., Bobed, C.: Meg: Medical knowledge-augmented large language models for question answering. arXiv preprint arXiv:2411.03883 (2025). `https://doi.org/10.48550/arXiv.2411.03883`
6. Chen, C., Leitch, A.: Llms as academic reading companions: Extending hci through synthetic personae. arXiv preprint arXiv:2403.19506 (2024). `https://doi.org/10.48550/arXiv.2403.19506`

7. Efaz, M.: MeReader: Progress-Aware, On-Device Reading Companion. GitHub repository (2025), https://github.com/WhiteHades/mereader
8. Faruq, O.: Assistant: AI helper plugin for KOReader. https://github.com/omer-faruq/assistant.koplugin (2026), gitHub repository
9. Gao, Y., et al.: Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997 (2024). https://doi.org/10.48550/arXiv.2312.10997
10. George Orwell: Nineteen eighty-four, accessed: Nov. 04, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Nineteen_Eighty-Four
11. Ippolito, D., et al.: Preventing verbatim memorization in language models gives a false sense of privacy. arXiv preprint arXiv:2210.17546 (2023). https://doi.org/10.48550/arXiv.2210.17546
12. Jiang, Z., et al.: Active retrieval augmented generation. arXiv preprint arXiv:2305.06983 (2023). https://doi.org/10.48550/arXiv.2305.06983
13. John Steinbeck: Of mice and men, accessed: Nov. 04, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Of_Mice_and_Men
14. Karpukhin, V., et al.: Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906 (2020). https://doi.org/10.48550/arXiv.2004.04906
15. KOReader Contributors: KOReader: An ebook reader application supporting pdf, djvu, epub, fb2 and many more formats. https://github.com/koreader/koreader (2026), gitHub repository
16. Kryściński, W., Rajani, N., Agarwal, D., Xiong, C., Radev, D.: Booksum: A collection of datasets for long-form narrative summarization. arXiv preprint arXiv:2105.08209 (2022). https://doi.org/10.48550/arXiv.2105.08209
17. Leo Tolstoy: The death of ivan ilyich, accessed: Nov. 04, 2025. [Online]. Available: https://en.wikipedia.org/wiki/The_Death_of_Ivan_Ilyich
18. Levonian, Z., et al.: Retrieval-augmented generation to improve math question-answering: Trade-offs between groundedness and human preference. arXiv preprint arXiv:2310.03184 (2023). https://doi.org/10.48550/arXiv.2310.03184
19. Mireshghallah, F., Uniyal, A., Wang, T., Evans, D., Berg-Kirkpatrick, T.: An empirical analysis of memorization in fine-tuned autoregressive language models. In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1816–1826 (2022). https://doi.org/10.18653/v1/2022.emnlp-main.119
20. Mitra, C., Miroyan, M., Jain, R., Kumud, V., Ranade, G., Norouzi, N.: Retllm-e: Retrieval-prompt strategy for question-answering on student discussion forums. Proceedings of the AAAI Conference on Artificial Intelligence **38**(21) (2024). https://doi.org/10.1609/aaai.v38i21.30368
21. Nomic AI: Nomic Embed Text V2: An Open Source, Multilingual, Mixture-of-Experts Embedding Model, accessed: Jun. 04, 2025. [Online]. Available: https://www.nomic.ai/blog/posts/nomic-embed-text-v2
22. Ollama: Ollama, accessed: Jun. 04, 2025. [Online]. Available: https://ollama.com
23. OpenAI: ChatGPT, accessed: May 29, 2025. [Online]. Available: https://openai.com/chatgpt/overview/
24. Preston, D.: Kindle app now answers questions about the book you're reading. The Verge (Dec 2025), https://www.theverge.com/news/844538/kindle-app-ask-this-book-ai-ios
25. Qdrant: Qdrant - Vector Database, accessed: Jun. 04, 2025. [Online]. Available: https://qdrant.tech/

26. Qu, Y., et al.: Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2010.08191 (2021). https://doi.org/10.48550/arXiv.2010.08191
27. Rakuten Kobo: Kobo, accessed: May 27, 2026. [Online]. Available: https://www.kobo.com/
28. Reichman, B., Heck, L.: Dense passage retrieval: Is it retrieving? arXiv preprint arXiv:2402.11035 (2024). https://doi.org/10.48550/arXiv.2402.11035
29. Robertson, S., Zaragoza, H., Taylor, M.: Simple bm25 extension to multiple weighted fields. In: Proceedings of the thirteenth ACM international conference on Information and knowledge management. pp. 42–49 (2004)
30. ScienceDirect: Pareto frontier - an overview, accessed: Sep. 10, 2025. [Online]. Available: https://www.sciencedirect.com/topics/engineering/pareto-frontier
31. Spatharioti, S.E., Rothschild, D.M., Goldstein, D.G., Hofman, J.M.: Comparing traditional and llm-based search for consumer choice: A randomized experiment. arXiv preprint arXiv:2307.03744 (2023). https://doi.org/10.48550/arXiv.2307.03744
32. SQLite: SQLite Home Page, accessed: Jun. 04, 2025. [Online]. Available: https://www.sqlite.org/
33. Wang, X., et al.: Searching for best practices in retrieval-augmented generation. arXiv preprint arXiv:2407.01219 (2024)
34. Weatherbed, J.: Adobe Acrobat gets a generative AI assistant to chat with your documents. The Verge (Feb 2024), https://www.theverge.com/2024/2/20/24077217/adobe-acrobat-generative-ai-assistant-chatbot-pdf-document
35. Xu, J., et al.: On-device language models: A comprehensive review. arXiv preprint arXiv:2409.00088 (2024). https://doi.org/10.48550/arXiv.2409.00088
36. Zhao, W., Deng, Z., Yadav, S., Yu, P.S.: Heterogeneous knowledge grounding for medical question answering with retrieval augmented large language model. In: Companion Proceedings of the ACM Web Conference 2024 (WWW '24). pp. 1590–1594. ACM (2024). https://doi.org/10.1145/3589335.3651941
37. Zhou, H., Boovaraghavan, S., Goel, M., Agarwal, Y.: On-device speech filtering for privacy-preserving acoustic activity recognition. In: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24). pp. 1802–1804 (2024). https://doi.org/10.1145/3636534.3698865