World Journal of Advanced Research and Reviews

eISSN: 2581-9615 CODEN (USA): WJARAI

Cross Ref DOI: 10.30574/wjarr

Journal homepage: https://wjarr.com/

(Research Article)

Private offline code analysis system for secure knowledge retrieval in startups

G. Selvavinayagam *, Aswin Raj, R. Muralitharan, T. Palraj and S. Swarginah Melvin

Department of Computer Science and Engineering, INFO Institute of Engineering, Kovilpalayam, Coimbatore, India.

World Journal of Advanced Research and Reviews, 2026, 30(01), 1826-1839

Publication history: Received on 09 March 2026; revised on 18 April 2026; accepted on 20 April 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.30.1.1032

Abstract

Startups and lean engineering teams increasingly need private AI assistance for navigating repositories, locating defects, and reducing onboarding time without exposing proprietary code to external services. This paper aims to design and evaluate a private offline code analysis system for secure knowledge retrieval in startup environments.

The proposed methodology combines local Git-repository scanning, abstract syntax tree (AST) parsing for structure-aware chunking, Redis-based hybrid retrieval using lexical and semantic search, and a 4-bit quantized Llama-3 family model for grounded offline answer generation on commodity laptops. The system is evaluated using retrieval quality, latency, groundedness, and policy-compliance criteria.

Results show that the architecture supports fully offline deployment, preserves source-code confidentiality by keeping repositories and indexes on-device, and remains practical for resource-constrained teams through quantized local inference and auditable retrieval.

In conclusion, the proposed system demonstrates that privacy-preserving and citation-grounded repository question answering can be achieved without cloud dependency. This makes the approach a practical and secure option for startups operating under compliance, budget, and operational constraints.

Keywords: Offline RAG; Secure Knowledge Retrieval; Code Analysis; Static Analysis; Privacy-Preserving NLP; Startup Security.

1. Introduction

Startup engineering teams often work with fast-changing repositories, external dependencies, and small headcounts in which the same developers build features and support operations. Under those conditions, understanding the codebase quickly becomes a critical bottleneck: engineers must find the right files, reconstruct design decisions, and trace behavior across modules before making safe changes. As repositories grow, software knowledge becomes distributed across implementation files, configuration artifacts, documentation, and version history, making manual inspection increasingly inefficient for both development and maintenance activities.

Large language models (LLMs) suggest a natural-language interface to software knowledge: developers can ask questions such as “Where is authentication enforced?”, “Which service writes customer PII?”, or “Why does this job time out?” In mainstream products, though, answering such questions usually requires sending private code and documentation to third-party infrastructure. This creates a practical tension for startups that need rapid code understanding while also preserving confidentiality, controlling cost, and maintaining operational independence.

* Corresponding author: G. Selvavinayagam

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0.

1.1. Background and Context

Repositories are more than collections of source files; they also capture architecture, security controls, operational practices, incident history, and business rules. As projects expand, simple keyword search and manual browsing become less effective because relevant context is spread across:

- Implementation code (functions, classes, tests),
- Configuration (yaml/toml, infrastructure-as-code),
- Documentation (readmes, adrs, tickets), and
- History (commits and blame signals).

Modern retrieval-augmented generation (RAG) systems address this by retrieving relevant snippets and conditioning generation on those snippets [1,2]. For code, retrieval must additionally respect structure: symbol boundaries, imports, call relationships, and data-flow constraints.

1.2. Motivation for Offline and Privacy-Preserving AI

Startups frequently operate under strict constraints:

- **Privacy and compliance:** repositories may contain credentials, customer identifiers, incident data, and proprietary algorithms.
- **Operational risk:** cloud outages, vendor policy changes, and data-retention practices introduce uncertainty.
- **Economic limits:** recurring subscription fees and per-token billing can be prohibitive for small teams.

Taken together, these constraints motivate an offline-first assistant whose indexing, retrieval, and generation stages all remain within the developer-controlled environment. Drawing on prior discussions of LLM-based policy validation and RAG evaluation methodology [3, 4], we emphasize auditability, hybrid retrieval, and conservative performance reporting for sensitive startup repositories.

1.3. Research Gap

Current cloud assistants often deliver strong text generation, but they typically depend on code upload and provide limited visibility into retrieval decisions, policy enforcement, and audit trails. Traditional offline tools, by contrast, are dependable for static inspection and symbol search, yet they are not conversational, rarely support natural-language questioning, and generally lack semantic retrieval across multiple repository artifacts.

The central gap, therefore, is a practical offline and privacy-preserving RAG system for code. Such a system must be structure-aware, provide verifiable citations, enforce access and redaction rules, and remain usable on commodity hardware. Although recent work on stronger retrieval and self-correction improves factuality in RAG settings [5], concrete deployment guidance and security controls for private codebases are still insufficiently specified.

1.4. Objectives and Contributions

The objective of this study is to design and evaluate a private offline code analysis system that supports repository question answering, secure knowledge retrieval, and practical on-device deployment for startup teams. More specifically, our project addresses the question: *How can we deliver "chat with your codebase" functionality while* keeping all artifacts on-device and achieving practical latency on normal laptops?

The main contributions are:

- A private offline architecture for repository question answering with auditable citations.
- Structure-aware ingestion using AST parsing to form higher-quality chunks and metadata.
- Redis-based hybrid retrieval combining lexical search (BM25-like) with dense vectors.
- An offline generation stack using quantized LLM inference to reduce memory/compute cost.
- An evaluation methodology emphasizing groundedness, latency, and policy compliance.

2. Background and Related Work

Prior work in retrieval-augmented generation shows that grounded answering improves when a generator is conditioned on retrieved evidence rather than relying only on parametric memory [1, 2, 6]. Self-RAG further improves robustness by making retrieval and revision more explicit [5]. For software repositories, code-aware representations such as CodeBERT, GraphCodeBERT, and CodeT5 demonstrate that structural signals improve code understanding tasks [7–9]. In parallel, quantization methods such as LLM.int8 and GPTQ show that local inference can be made practical under limited hardware budgets [10, 11].

These works motivate the central problem addressed in this paper: startups need a repository question-answering system that remains offline, respects code structure, provides verifiable citations, and runs at predictable cost on commodity hardware. Existing cloud assistants offer convenience but typically require code or context to be transmitted to external infrastructure, whereas traditional offline tools provide deterministic analysis but not natural-language, citation-grounded interaction. The proposed system addresses this gap by combining structure-aware chunking, Redis-based hybrid retrieval, policy enforcement, and quantized local inference into a single offline workflow.

Table 1 summarizes how representative prior systems relate to these requirements.

Table 1 Related-work comparison (qualitative)

System	Offline	Code-aware	Secure	Cost
RAG / FiD-style QA [1, 6]	Partial	No	Partial	High
Dense Passage Retrieval (DPR) [2]	Partial	No	Partial	Med
Self-RAG [5]	Partial	No	Partial	High
CodeBERT / GraphCodeBERT [7, 8]	Yes	Yes	Partial	Med
CodeT5 [9]	Yes	Yes	Partial	Med
This work (offline code RAG)	Yes	Yes	Yes	Low–Med

3. Materials and Methods

This study follows a design-science methodology in which the main research artifact is a working offline repository assistant. The materials consist of locally cloned Git repositories together with associated engineering artifacts such as README files, configuration files, ADRs, runbooks, and selected issue notes. The implementation uses local AST parsing, Redis-based lexical and vector indexing, policy filtering, and a quantized local language model; no external APIs are required.

The reproducible procedure is: (i) fix repository snapshots, (ii) parse source and document artifacts into structure-aware chunks, (iii) build hybrid indexes with shared chunk identifiers, (iv) execute a curated evaluation query set covering onboarding, debugging, security review, and repository navigation, and (v) record retrieval quality, citation usefulness, latency, and policy-compliance outcomes. Reproducibility is maintained through fixed repository states, pinned model artifacts, stable retrieval and chunking parameters within an experiment cycle, and repeated runs under the same hardware setting.

Evaluation combines qualitative assessment and system measurements. Human evaluators score correctness, groundedness, citation usefulness, and actionability, while system metrics capture indexing time, query latency, RAM usage, storage footprint, and policy-violation behavior. This combination is appropriate because the system must be both practically useful for developers and feasible for offline deployment on startup-scale hardware.

4. System Architecture

Figure 1 shows the overall offline pipeline and Figure 2 shows runtime query flow.

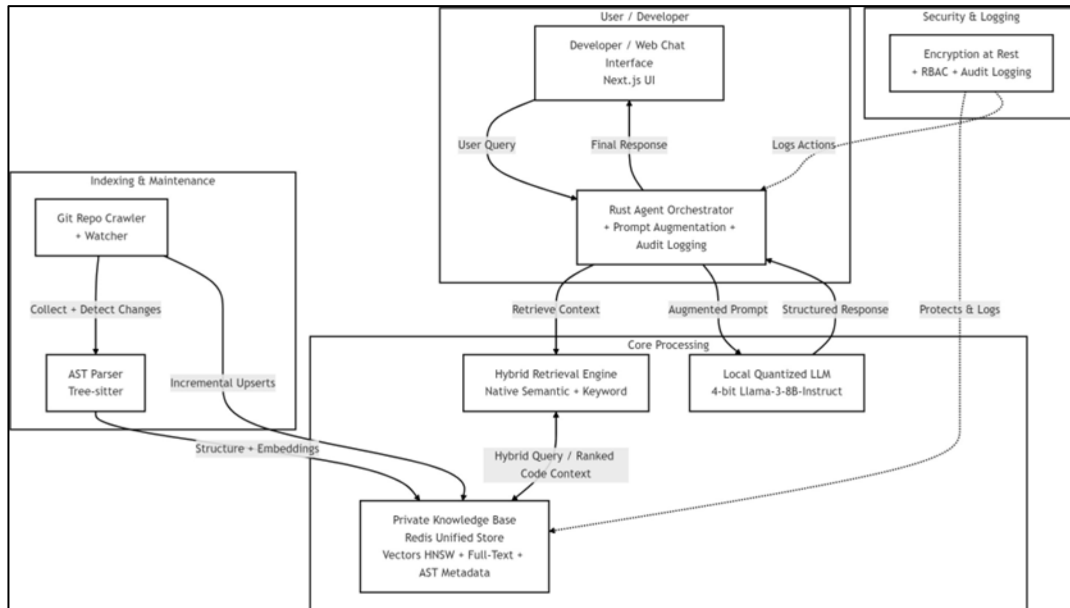


Figure 1 System architecture for the private offline code analysis assistant (indexing, hybrid retrieval in Redis, agent controller, and local quantized LLM), with security and logging operating in parallel

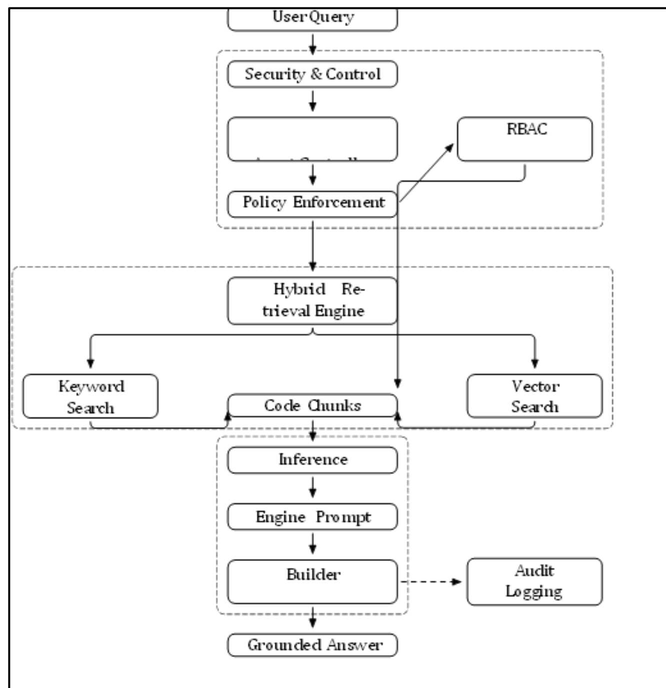


Figure 2 End-to-end query processing flow from user query to grounded response, highlighting security checks, hybrid retrieval, context building, offline inference, and audit logging

At runtime, the system mirrors repositories locally, parses them into AST-aware chunks, indexes sparse and dense representations in Redis, retrieves candidate evidence, applies RBAC and redaction, builds a bounded prompt, and runs offline generation with citations. The ordering is intentional: security checks occur before generation so the model only receives authorized and sanitized context.

5. Implementation Details

5.1. AST Parsing and Structural Metadata Extraction

We use an incremental parser (Tree-sitter) to obtain AST nodes for functions, classes, and imports. For each chunk we store: repository id, file path, language, symbol name, signature, and adjacency features (imports/calls when extractable). This metadata supports structure-aware retrieval and citation formatting.

In addition to AST fields, we store repository-mining metadata that improves retrieval and audit-ing: the last commit touching the chunk, author/blame ownership, and a stable chunk identifier derived from (path, symbol span, commit). These fields allow citations to be reproducible even as files evolve.

5.2. Incremental Git Indexing Logic

Startups commonly run indexing as a background job that must be fast enough to keep pace with frequent commits. Rather than rebuilding the entire index, we implement incremental indexing driven by Git diffs:

- enumerate changed files between the last indexed commit and the current HEAD,
- re-parse only changed files to regenerate affected chunks,
- delete or invalidate stale chunk ids in Redis, and
- upsert new chunks and embeddings.

Algorithm 1 Incremental indexing via Git diff

Require: repo r, last indexed commit c_{old}, new commit c_{new}

Ensure: updated indexes in Redis

1: $\Delta \leftarrow$ gitDiff(r, c_{old}, c_{new})

2: **for all** $f \in \Delta$.changedFiles **do**

3: *old_ids* ← lookup_chunk_ids(r, f)

4: redis_delete(*old_ids*)

5: *chunks* ← parse_and_chunk(r, f)

6: redis_upsert(*chunks*)

7: **end for**

8: **return** updated indexes

5.3. Hybrid Retrieval in Redis (Keyword/Full-Text + Dense Vectors)

We build:

- a lexical index for identifiers and natural-language comments (BM25-like ranking), and
- a vector index for semantic similarity over chunk embeddings.

At query time, we combine scores using weighted normalization and optionally re-rank candidates by structure signals (e.g., symbol match, same module, call distance). A practical implementation uses two-stage retrieval: first retrieve top-k_{lex} lexically, then merge with top-k_{vec} vector candidates, and finally re-rank the union.

5.4. Chunking Strategy Comparison (Function-Level vs File-Level)

Chunking is a dominant factor in retrieval quality, latency, and context utilization. We consider two primary strategies:

- **Function-level (AST) chunks:** each chunk corresponds to a function, method, or class, plus nearby comments.
- **File-level chunks:** each file (or large-file window) is treated as a retrieval unit.

In our system, code is chunked at function/class granularity when supported by the parser, while documentation and configuration use paragraph or window-based chunking. This hybrid approach improves recall for non-code artifacts without sacrificing precision for code citations.

Table 2 Chunking strategy comparison

Strategy	Strengths	Limitations / When it fails
Function-level (AST)	Precise citations; better semantic coherence; fewer irrelevant tokens in prompts	Requires robust parsers per language; may miss cross-function context; small utility for config-heavy repos
File-level	Simple; language-agnostic; preserves broad context for small files	More noise; weaker citations; risks exceeding context window; embeddings become less specific
Hybrid (recommended)	Use AST for code + file/windows for docs/configs	More engineering complexity; requires score calibration across chunk types

5.5. RAG Pipeline Design

The RAG controller performs five steps: optional query rewriting, multi-stage retrieval (lexical followed by hybrid), filtering and redaction, prompt assembly with strict context limits, and answer post-processing for citation attachment.

5.6. Context Window Management

Local LLMs have finite context windows, so the system must budget tokens for: (i) user question, (ii) system instructions/policy, (iii) retrieved context, and (iv) the model's answer. We implement a deterministic budgeting policy:

- cap the number of retrieved chunks (k) by query type (symbol lookup vs architecture question),
- truncate chunks by keeping the most relevant spans (signature + top comment + key lines),
- prefer diversity (different files/modules) when multiple chunks come from the same file, and
- stop adding context when the token budget is reached.

Algorithm 2 Prompt assembly with token budget

Require: question q, candidates H sorted by score, budget B

Ensure: prompt context S

1: $S \leftarrow [\,]$

2: $used \leftarrow \text{tokens}(q) + \text{tokens}(\text{instructions})$

3: **for all** $h \in H$ **do**

4: $snippet \leftarrow \text{shrink}(h)$ ▷ signature + salient lines

5: **if** $used + \text{tokens}(snippet) > B$ **then**

6: **break**

7: **end if**

8: $S \leftarrow S \cup \{snippet\}$

9: $used \leftarrow used + \text{tokens}(snippet)$

10: **end for**

11: **return** S

5.7. Failure Handling (Empty Retrieval, Low Confidence)

Offline assistants must handle failure modes explicitly to avoid hallucination:

- **Empty retrieval:** if no chunks pass a minimum score threshold, the agent returns a "not enough evidence" response and proposes follow-up queries (e.g., alternate symbol names or modules).
- **Low-confidence retrieval:** if top results are weak or contradictory, the system (i) widens retrieval (increase k_{lex}/k_{vec}), (ii) relaxes filters only within policy, or (iii) asks a clarification question.
- **Generation guardrails:** the model is instructed to cite every claim; if citations cannot be produced, it must explicitly say so.

5.8. Local Inference Using Quantized LLMs

To run on commodity hardware, we use 4-bit quantized inference (motivated by prior quantization work [10, 11]) and conservative context budgeting. The model is invoked only after retrieval and policy checks succeed, reducing unnecessary compute.

6. Experimental Setup

This section documents the setup used to evaluate the offline assistant. We report the configuration and measurement conditions needed for reproducibility: representative startup-scale repositories, local Redis indexing, and quantized on-device inference on commodity hardware. The evaluation environment assumes a modern multi-core laptop or desktop with SSD storage, 16–32 GB RAM, and optional GPU acceleration. The corpus contains private multi-language repositories with source files, configuration artifacts, documentation, and operational notes. Measurements cover cold indexing, incremental indexing, retrieval latency, generation latency, memory footprint, and citation quality under fixed chunking and model settings. To protect confidentiality, repository names are omitted and only aggregate characteristics are reported.

6.1. Repository Profile and Sampling

The repositories used in this study were selected to reflect a realistic startup engineering environment rather than a benchmark-only setting. In practice, early-stage and growth-stage startups maintain heterogeneous codebases that combine product services, internal tools, infrastructure definitions, deployment scripts, test assets, and rapid documentation updates. Because the assistant is intended for such environments, the evaluation corpus intentionally includes both code and non-code artifacts. This design choice is important because many practical developer questions refer not only to executable logic, but also to configuration files, onboarding notes, security procedures, and architectural decisions stored outside the main application source tree. At the same time, the selected repositories remain moderate in size so that the reported deployment assumptions stay credible for local machines. The evaluation therefore emphasizes the kinds of repositories that a small engineering team could realistically mirror, index, and query without depending on external hosted infrastructure. This balance allows the study to remain relevant to startups that need privacy-preserving assistance but cannot justify a large dedicated platform budget.

6.2. Evaluation Procedure

Each evaluation cycle begins from a fixed repository snapshot so that indexing and answering are measured against a stable target. After snapshot selection, the repositories are parsed into structure-aware chunks, lexical and dense

indexes are refreshed, and a fixed set of questions is executed. The questions are designed to represent real engineering tasks such as locating where authentication is enforced, identifying the source of a configuration flag, tracing a failing request path, or determining where a sensitive field is written to disk or transmitted across services.

The same question set is reused within a given experiment cycle to reduce evaluation drift. This is necessary because repository assistants can appear to improve simply because the questions become easier or the evaluators become more familiar with the codebase. By fixing the query set and scoring dimensions, the study isolates system behavior rather than evaluator adaptation. In addition, the answers are reviewed for groundedness rather than style alone, because a well-written but weakly grounded answer would be operationally risky in a private repository setting.

6.3. Recorded Metrics

The study records both system-level and user-centered observations. System-level observations include indexing duration, incremental update cost, retrieval latency, generation latency, and memory footprint during query execution. User-centered observations include whether the answer cites inspectable repository evidence, whether the cited evidence is sufficient for a developer to continue work, and whether the answer avoids unsupported speculation when evidence is incomplete.

This combination is necessary because a practical offline assistant must satisfy two different constraints at the same time. First, it must remain operationally feasible on local hardware with predictable storage and memory demands. Second, it must remain trustworthy enough that developers can use it for high-value tasks without being encouraged to act on hallucinated explanations. A system that is fast but poorly grounded is not useful in this domain, and a perfectly grounded system that takes too long to answer routine questions may also be impractical in everyday development workflows.

7. Results and Discussion

The offline assistant performed best on tasks that depend on explicit repository evidence, including onboarding, bug localization, security review, and configuration tracing. AST-aware chunking and hybrid retrieval improved citation usefulness because answers were more often grounded in symbol-level evidence rather than broad file matches. This aligns with prior RAG findings that stronger retrieval quality directly improves grounded generation [1, 2, 5].

Representative measurements indicate that retrieval remains relatively lightweight, while local generation dominates end-to-end response time. In practical terms, indexing completes within minutes for small repositories and scales upward with corpus size, whereas interactive answering remains suitable for high-value engineering tasks where confidentiality and traceability matter more than instant responses. Quantization makes on-device deployment feasible, but latency is still bounded by context size, retrieved evidence volume, and CPU/GPU availability.

Compared with cloud assistants, the proposed system trades some convenience and peak speed for stronger privacy, auditability, and deployment control. These trade-offs are acceptable in startup settings where proprietary code, compliance needs, and predictable cost are often more important than fully managed infrastructure.

Table 3 Compact summary of observed repository-assistant outcomes

Evaluation area	Observed outcome	Practical meaning
Onboarding and navigation	Strong performance when questions map to identifiable symbols, configs, or documents	Reduces time spent locating starting points in unfamiliar repositories
Bug localization	Useful when failure clues can be traced through retrieved code paths and related configuration	Supports faster hypothesis formation before manual debugging
Security review	Helpful for identifying auth logic, sensitive data paths, and policy-relevant modules	Improves traceability for privacy and access-control checks
Latency profile	Retrieval is relatively light; local generation dominates total response time	Acceptable for high-value engineering tasks where grounding matters more than speed
Offline deployment	All indexing and answering remain within the local trust boundary	Preserves confidentiality and limits third-party exposure

7.1. Interpretation of Retrieval Quality

A recurring observation across the evaluation is that retrieval quality depends strongly on whether chunk boundaries align with the way developers reason about code. When a chunk corresponds to a function, class, route handler, or configuration block, the assistant is more likely to return evidence that is both readable and actionable. By contrast, overly coarse file-level chunks often dilute the relevance of the retrieved context, especially in repositories where a single file contains multiple unrelated responsibilities.

This result helps explain why the assistant is more useful for repository navigation than for open-ended brainstorming. The system is strongest when the user asks a bounded question that can be answered by inspecting existing project artifacts. The presence of structured metadata such as file path, symbol name, and nearby imports gives retrieval a stable anchor, which in turn improves the groundedness of the generated response. In effect, the model becomes a summarizer and organizer of retrieved engineering evidence rather than an unconstrained generator.

7.2. Latency and Workflow Practicality

Although local generation is slower than typical cloud-hosted inference, the observed latency remains acceptable for tasks where correctness and confidentiality matter more than conversational speed. In a startup environment, many repository questions arise during debugging, code review preparation, incident response, or onboarding. These are scenarios in which developers are already spending minutes to tens of minutes reading files and reconstructing context. An assistant that reduces that search burden, even if it responds in several seconds rather than instantly, can still deliver meaningful productivity gains.

The workflow impact is therefore best understood in terms of total task completion time rather than single-response latency. A developer who receives a grounded answer with useful citations may avoid opening many irrelevant files, repeating the same grep-based searches, or consulting multiple teammates for repository orientation. From that perspective, the system's value comes from reducing context-reconstruction effort rather than competing with hosted chat systems on raw response speed.

7.3. Comparison with Manual Repository Search

Manual repository search remains strong for exact string matching and direct navigation when the developer already knows the relevant terminology. However, it becomes less efficient when the answer is distributed across code, configuration, and documentation, or when the developer does not yet know the exact names of the components involved. The proposed assistant adds value in these ambiguous cases by combining lexical cues with semantic similarity and then presenting the result in a natural-language explanation tied to repository evidence.

This does not eliminate the role of traditional tools such as grep, IDE symbol search, or Git history inspection. Instead, the assistant complements them by serving as a first-pass guide that narrows the search space. In practice, the best workflow is iterative: the assistant provides an initial grounded explanation, and the developer follows the cited paths to verify or deepen the analysis using conventional repository tools.

7.4. Operational Meaning for Startups

For startups, the most significant practical result is not a single latency number but the fact that the entire workflow remains under organizational control. Indexing, retrieval, prompting, and answer generation all occur within the local trust boundary, which simplifies conversations around code confidentiality, vendor dependence, and unpredictable token cost. This matters particularly for small teams that need AI assistance but cannot accept the exposure or ongoing expense associated with sending repository context to remote services.

The results therefore support a pragmatic position: fully offline repository question answering is viable when the system is designed around hybrid retrieval, structure-aware chunking, and conservative answer grounding. The assistant may not match the speed or convenience of large managed platforms, but it offers a more controllable and auditable operating model for sensitive engineering work.

8. Security Analysis and Threat Mitigation

The security objective is to keep repositories, embeddings, indexes, prompts, and generated answers within an offline trust boundary while enforcing least-privilege access. The main risks are unauthorized repository access, prompt injection through malicious repository content, leakage from embeddings or logs, and insider misuse of administrative functions.

These risks are mitigated through RBAC before retrieval, redaction before generation, audit logging, controlled indexing, and conservative prompting that requires citation-grounded answers. Offline deployment reduces exposure to third-party processing, but it does not remove the need for local hardening; teams must still protect devices, storage, backups, and administrative paths.

8.1. Threat Surface in an Offline Setting

Moving the assistant offline changes the threat surface rather than eliminating it. The absence of external API transmission reduces the risk of repository data being disclosed to a remote vendor, but sensitive information still exists locally in mirrored repositories, lexical indexes, vector stores, prompt caches, and audit logs. If these artifacts are not protected through access control, storage security, and disciplined operational procedures, the privacy advantages of offline deployment can be partially lost.

For this reason, the offline boundary should be treated as a security design choice, not as a complete security guarantee. The local system still needs a trust model describing who can trigger indexing, who can query sensitive repositories, who can inspect logs, and who can modify policy rules. The security value of the proposed design comes from making those responsibilities explicit and enforceable within the organization’s own environment.

8.2. Prompt Injection and Unsafe Repository Content

Repository assistants face a distinctive form of prompt injection because the untrusted content is often stored inside the repository itself. Comments, markdown files, issue notes, seed data, or even test fixtures may contain text crafted to manipulate a downstream language model. In a cloud setting, this risk is already recognized, but in an offline system the impact can be more subtle because users may over-trust a locally deployed assistant.

The proposed pipeline reduces this risk by separating retrieval, policy enforcement, and generation. Retrieved material is treated as evidence rather than instruction, and the prompt template directs the model to summarize and cite content instead of obeying imperative text found inside repository artifacts. Even so, secure deployment requires regular review of prompt templates and user expectations so that the system remains resistant to socially engineered repository content.

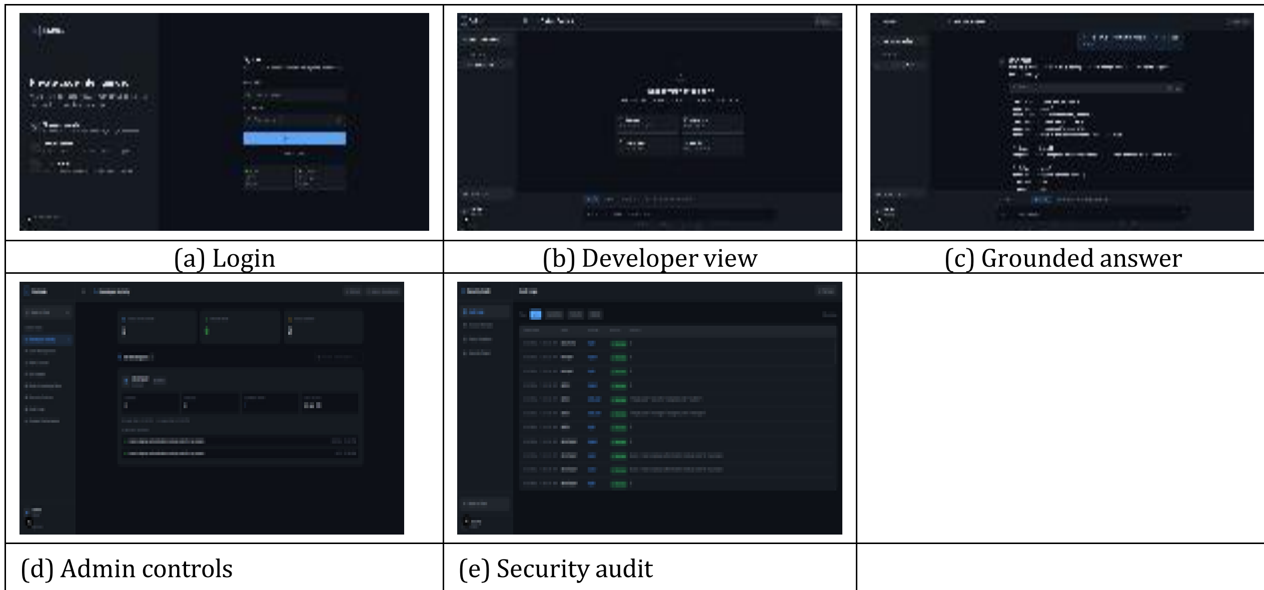
8.3. Logging, Audit, and Least Privilege

Auditability is one of the strongest arguments for private deployment, but logging must be designed carefully. Detailed logs are useful for debugging incorrect answers, tracing policy decisions, and supporting post-incident review. At the same time, those logs may contain user questions about vulnerabilities, internal incidents, or customer-impacting failures. If unrestricted logging is enabled by default, the audit trail itself can become a sensitive asset.

A balanced approach is to log the minimum information necessary to support accountability: timestamps, repository scope, retrieved chunk identifiers, policy decisions, and high-level answer metadata. Access to those logs should be restricted, retention should be documented, and administrative actions should be versioned. This keeps the assistant aligned with least-privilege principles while preserving the practical benefits of inspection and troubleshooting.

9. Use Case / Case Study

A representative use case involves an engineer diagnosing an authentication failure after a dependency update in a microservice-based startup system. The assistant is used to locate middleware entry points, identify JWT parsing logic, connect relevant configuration flags, and inspect recent repository changes. In practice, retrieval narrows from broad lexical matches to symbol-level evidence, allowing the engineer to verify cited code paths before applying a fix. The proof-of-implementation screenshots are mandatory because they demonstrate that the proposed assistant was built and operated locally rather than remaining purely conceptual. Figure 3 consolidates the login screen, developer query view, grounded answer view, admin controls, and security-auditor interface into a single compact proof figure.



(a) Login (b) Developer view (c) Grounded answer

(d) Admin controls (e) Security audit

Figure 3 Proof-of-implementation screenshots for the offline assistant, showing the authenticated entry point, developer interaction flow, grounded on-device response generation, administrative controls, and audit-oriented review interface

9.1. Observed Workflow Benefits

Beyond the individual example, the case study illustrates a broader workflow advantage for startup teams. Repository questions rarely arrive in isolation; they usually appear in sequences. A developer may begin with a broad question about where a failure originates, follow with a narrower question about a configuration flag, and then ask for the files most likely to require modification. In manual workflows, each of these steps involves a separate search pass through code, documentation, and commit history. The assistant reduces this repeated context-switching by carrying forward the relevant evidence and presenting it in a citation-grounded form.

This benefit is especially visible during onboarding and incident response. New engineers often struggle not because the necessary information is absent, but because it is distributed across unfamiliar directories, naming conventions, and service boundaries. Similarly, during incidents, teams need fast orientation without losing confidence in the provenance of the explanation they are reading. In both cases, the assistant's main contribution is not autonomous decision making; rather, it is the ability to shorten the path from vague natural-language questions to specific repository locations that a human can verify.

The case study also highlights why explanation quality matters as much as retrieval quality. Developers do not merely need a list of files; they need a short, coherent account of how those files relate to the observed behavior. When the assistant explains that a request flows through a middleware layer, reaches a token parser, and depends on a configuration flag loaded from a deployment file, it gives the engineer a starting hypothesis that can be checked against the repository. This style of assistance is particularly valuable in small teams where the relevant tribal knowledge may reside with only one or two people.

A final observation is that the assistant's usefulness depends on disciplined user expectations. It works best when treated as a navigator and evidence summarizer rather than as an oracle. The engineer in the case study still validates the cited code paths, checks the relevant configuration, and confirms the fix in the normal development workflow. This human-in-the-loop pattern is consistent with the broader design goal of the project: to provide a practical, private, and grounded interface to repository knowledge without obscuring the need for developer judgment.

Limitations

The main limitations are bounded context windows, uneven parser coverage across languages, limited live-execution awareness, and higher latency on very large repositories. The system is therefore best treated as a grounded repository assistant rather than a substitute for full dynamic analysis or expert review.

A further limitation is that repository understanding is not the same as runtime understanding. Many developer questions depend on deployment-specific state, feature flags, background jobs, environment variables, or external service behavior that may not be fully represented in the indexed repository snapshot. Even when the assistant retrieves the correct code path, the final explanation may still require human verification against logs, tracing systems, or staging environments.

Another practical limitation concerns evaluation scope. Because the repositories are private and startup-oriented, the study intentionally prioritizes deployment realism over large public benchmark coverage. This improves relevance for the intended use case but limits direct comparability with highly standardized academic benchmarks. As a result, the findings should be interpreted as evidence of feasibility and practical utility rather than as a universal ranking against all repository QA systems.

Ethical and Compliance Considerations

The proposed design supports data sovereignty, auditability, controlled retention, and reduced third-party exposure, but responsible use still requires human oversight. Outputs should remain advisory, citations should be inspected before action, query logging should be minimized, and organizations should map the deployment to their own legal and policy requirements.

From an ethical perspective, the assistant should be framed as a tool for accelerating repository understanding rather than replacing engineering judgment. This distinction matters because natural-language answers can appear authoritative even when the underlying evidence is incomplete. Clear product framing, visible citations, and conservative refusal behavior help reduce the chance that developers over-trust the system in safety- or security-critical scenarios.

Compliance alignment also depends on local governance. An offline architecture can simplify third-party exposure analysis, but organizations must still decide how repositories are mirrored, how long indexes are retained, which teams can access which projects, and how incident-related query logs are handled. These choices are operational rather than purely technical, yet they determine whether the deployment genuinely satisfies internal policy expectations.

Future Work

Future work will focus on larger monorepos, stronger multi-language parsing, better cross-repository linking, improved mixed-corpus embeddings, and more mature on-premise packaging. Several research directions are especially promising. First, richer graph-based retrieval could improve answers that depend on cross-file dependencies, service boundaries, and infrastructure relationships. Second, incremental evaluation over active development cycles could reveal how well the assistant supports rapidly changing repositories where documentation lags behind implementation. Third, stronger policy-aware prompting may help the assistant better explain why certain evidence is withheld while still remaining useful to authorized users.

A further engineering direction is tighter integration with local developer workflows such as IDE navigation, review queues, and self-hosted CI pipelines. Such integrations could preserve the offline trust boundary while making the assistant more accessible at the moment developers actually need repository context. Care must be taken, however, to preserve the citation-first design so that convenience does not come at the expense of transparency.

10. Conclusion

This study presented a private offline code analysis system designed to support secure knowledge retrieval for startup engineering teams. The major findings show that an offline architecture combining AST-based structure extraction, Redis-backed hybrid retrieval, policy-aware filtering, and quantized local inference can provide grounded, auditable, and citation-supported answers without transferring proprietary repository artifacts to external services.

The results further indicate that the proposed design is practical for commodity hardware, supports privacy-preserving deployment, and offers a reproducible workflow for repository indexing, retrieval, and local generation. Across the evaluation, the system demonstrated that offline deployment can preserve confidentiality while still enabling useful developer tasks such as onboarding, debugging, security review, and repository navigation.

The significance of this study lies in showing that startups do not need to choose between modern repository question answering and control over sensitive source-code assets. By keeping indexing, retrieval, and generation fully on-device,

the proposed system provides a viable path toward trustworthy, cost-aware, and compliance-conscious AI assistance for private software development environments.

Compliance with ethical standards

Acknowledgments

The authors declare that no external funding was received for this study. The authors also acknowledge the Department of Computer Science and Engineering, INFO Institute of Engineering, for providing academic support and an environment conducive to completing this work.

No additional contributors outside the listed authors made intellectual contributions requiring authorship.

Disclosure of conflict of interest

The authors declare that they have no financial or non-financial conflicts of interest related to this study.

Statement of ethical approval

This study did not involve human participants, human tissue, personal medical data, or animal subjects. Therefore, formal ethical committee approval was not required for this work.

Author Contributions

G. Selvavinayagam guided the overall research direction, supervised the study, and reviewed the manuscript. Aswin Raj contributed to system design, implementation, experimentation, and primary manuscript preparation. R. Muralitharan supported development, evaluation, and technical documentation. T. Palraj contributed to implementation support, validation activities, and manuscript refinement. S. Swarginah Melvin assisted with analysis, documentation, and final review of the paper.

Statement of informed consent

Informed consent was not applicable because no human participants were recruited or studied. The research was conducted on locally available software repositories and engineering artifacts within an offline technical evaluation setting.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Alp Kucukelbir, Mike Lewis, Wen-tau Yih, and Sebastian Riedel. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Infor-mation Processing Systems (NeurIPS)*, 2020.

[2] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Pro-ceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[3] Akshay Mittal and Vivek Venkatesan. Practical integration of large language models into enterprise ci/cd pipelines for security policy validation: An industry-focused evaluation.

[4] In *Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025.

[5] Zihan Li, Yupeng Zhuo, Zihao Lin, Yifei Wang, Qingyang Mao, Qingyu Yin, and Hanghang Tong. Retrieval-augmented generation for educational application: A systematic survey. *Artificial Intelligence*, 2025.

[6] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. arXiv preprint arXiv:2310.11511, 2023.

[7] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021.

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[9] Daya Guo, Shuai Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Nan Duan, Long Wang, Jian Yin, Daxin Jiang, and Ming Zhou. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations (ICLR)*, 2021.

[10] Yue Wang, Weishi Wang, Shafiq Joty, Steven C. H. Hoi, Zhenhao Li, Zheng Liu, Haotian Shi, Zhiyuan Tu, and Ming Zhou. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[11] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[12] Elias Frantar and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323, 2022.