J-DA
JOURNAL
DEKSTOP APPLICATION

Received: 5 May 2026 | Revised: 3 Jun 2026 | Accepted: 5 Jun 2026 | Published: 6 Jun 2026 E-ISSN: 2961-9688, P-ISSN: 2961-9696

DOI: 10.59431/jda.v5i1.810

RESEARCH ARTICLE

Large Language Model Integration in Desktop Applications: A Systematic Literature Review

Deyidi Mokoginta [1*]

[1*] Department of Electrical Engineering, Universitas Teknologi Sulawesi Utara, Manado City, North Sulawesi Province, Indonesia.

Correspondence

[1*] Department of Electrical Engineering, Universitas Teknologi Sulawesi Utara, Manado City, North Sulawesi Province, Indonesia.

Email: deydi81@gmail.com

Funding information
Universitas Teknologi Sulawesi Utara.

Abstract

The integration of Large Language Models (LLMs) into desktop applications has accelerated sharply over the past several years, yet no prior systematic review has examined this deployment setting as a distinct research domain — separate from web-based or mobile environments, each carrying different architectural constraints, privacy expectations, and user interaction patterns. This study maps, analyzes, and synthesizes the available scientific evidence on LLM architectures adopted in desktop implementations, the technical barriers arising during integration, the measurable effects on User Experience (UX), and the relative effectiveness of competing deployment strategies. A Systematic Literature Review (SLR) was conducted following PRISMA 2020 reporting guidelines, with research questions structured using the PICOC framework. Systematic searches across seven digital databases — IEEE Xplore, ACM Digital Library, Scopus, Web of Science, Google Scholar, arXiv, and Semantic Scholar — covering 2019–2024 yielded 1,200 initial articles, narrowed to 42 final articles through layered selection and quality appraisal using a modified CASP rubric. GPT-4 and the LLaMA family emerged as the dominant architectures, with model selection driven primarily by parameter scale and privacy requirements. Computational constraints, response latency, and data security ranked as the most serious technical barriers, with Retrieval-Augmented Generation (RAG) and model quantization as the most common mitigations. LLM integration was associated with measurable gains in task efficiency and user satisfaction in 73.8% of reviewed studies, though output predictability and user trust remained persistent challenges. A hybrid deployment approach — combining local models with cloud-based API access — emerged as the most common practice among production implementations. Unlike existing reviews addressing LLM capabilities in general software engineering contexts, this study provides the first systematic map of LLM integration specifically within the desktop application domain, offering concrete guidance for developers working at the intersection of artificial intelligence and desktop software engineering.

Keywords
Large Language Model; Desktop Application; Systematic Literature Review; Local Deployment; Retrieval-Augmented Generation.

1 | INTRODUCTION

The past decade of artificial intelligence research has not moved steadily — it has lurched forward in sudden, disorienting leaps. Nowhere is this more visible than in the emergence of Large Language Models (LLMs) as a class of technology that has fundamentally altered how humans interact with computing systems. LLMs are Transformer-based models trained on billions of parameters using massive text corpora, enabling them to understand and generate natural language at a level of fluency that, until recently, was considered a distant benchmark. Wang *et al*. (2025) traced the evolution of language models from early statistical approaches through the modern LLM era, marking the emergence of GPT-3, GPT-4, PaLM, and LLaMA as significant milestones in that trajectory. Patel and Dholakiya (2025) added comparative depth by evaluating GPT-4o, Grok, Claude, and LLaMA against standard benchmarks including MMLU and MATH, assessing each across architecture, parameter scale, and task performance. Han *et al*. (2024) examined the infrastructure underpinning these models — the Transformer architecture as a structural foundation, alongside optimization and compression methods that allow large models to operate efficiently across diverse computational environments.

LLM adoption has long since escaped the research laboratory. Chkirbene *et al*. (2024) documented the spread of LLMs across finance, healthcare, legal services, and education, noting that growing model accessibility has been the primary driver of this expansion. Schillaci (2024) narrowed the lens to the enterprise context, cataloguing providers, integrators, and end users within the business ecosystem, and identifying conversational search, content generation, and basic workflow automation as the dominant use cases. Schillaci (2024) also flagged the growing trend toward local LLM hosting as an alternative to external API access — a response to the privacy and security constraints that make cloud-only deployment untenable in regulated sectors. That shift signals something more than a technical preference. It marks a broader move toward embedding LLM capabilities directly into local software infrastructure, including desktop applications.

Desktop software has become one of the more consequential frontiers for LLM integration in modern software development. Chkirbene *et al*. (2024) cited GitHub Copilot as a concrete example of LLM-driven change in desktop-based software development, where AI-assisted code generation operates within an integrated development environment. Zhang *et al*. (2024) extended this view by surveying LLM-based GUI agents operating across desktop operating systems, detailing how these agents interact with graphical interfaces to execute user instructions autonomously. Wang *et al*. (2024) identified key advances in LLM- and MLLM-based agent architectures capable of processing and interpreting GUI elements to support interaction patterns that resemble human behavior. On the deployment side, Zheng *et al*. (2025) provided technical depth on strategies for running LLMs on resource-constrained devices — covering efficient model design, quantization, pruning, and runtime inference optimization — all directly relevant to developers seeking local LLM integration without dependence on network connectivity.

Despite growing interest, systematic treatment of LLM integration specifically within the desktop application context remains thin. Hou *et al*. (2024) conducted a systematic review of 395 articles on LLMs in software engineering broadly, but that work did not treat desktop as a distinct deployment environment with its own technical characteristics and constraints. Scherbakov *et al*. (2025) showed that GPT- and ChatGPT-based LLMs dominate scientific review automation, yet their focus was methodological rather than concerned with technical integration into desktop software. Chkirbene *et al*. (2024) acknowledged that existing surveys tend to address LLM capabilities in general terms, without offering a unified evaluative framework for assessing integration quality in the desktop context — while real barriers such as high computational cost, privacy concerns, and limited explainability remain unmapped in any systematic way. That gap is the direct motivation for this study.

Accordingly, a Systematic Literature Review (SLR) was conducted using PRISMA 2020 methodology to map, analyze, and synthesize the scientific evidence on LLM integration in desktop applications. Four research questions guide the inquiry: (RQ1) Which LLM architectures are most commonly integrated into desktop applications? (RQ2) What are the principal technical challenges in integrating LLMs within desktop environments? (RQ3) How does LLM integration affect User Experience (UX) in desktop applications? (RQ4) Which deployment strategy — API-based or local — proves more effective for desktop LLM integration? The study's contributions include a systematic map of the LLM–desktop integration landscape, an inventory of reported technical challenges and solutions, and practical guidance for developers and researchers working at the intersection of artificial intelligence and desktop software engineering.

2 | BACKGROUND THEORY

2.1 Large Language Models: Definition, Evolution, and Architecture

Large Language Models are Transformer-based artificial intelligence systems trained on massive text corpora with parameter counts ranging from billions to trillions, enabling them to understand, generate, and manipulate natural

language with a high degree of fluency. Kukreja *et al*. (2024) described LLMs as Transformer-based models containing billions of parameters, with an architecture that allows the model to learn complex linguistic representations through large-scale pre-training. Mienye *et al*. (2025) added that the foundational LLM architecture rests on the Transformer, with two dominant variants: the decoder-only GPT (Generative Pre-trained Transformer) and the encoder-only BERT (Bidirectional Encoder Representations from Transformers), each suited to different natural language processing tasks. Choi and Chang (2025) confirmed that the Transformer architecture remains the central structural element across the entire modern LLM ecosystem, whether proprietary or open-source. The evolution of LLMs traces through a sequence of milestones, each building on the last. Wang *et al*. (2025) documented the long arc from early statistical language models to the modern era, noting that GPT-3's release with 175 billion parameters in 2020 marked a turning point in few-shot learning — the model's capacity to adapt to new tasks from minimal examples without retraining. GPT-4 then introduced multimodal capability, followed by GPT-4o, which unified text, image, and audio processing within a single architecture. Patel and Dholakiya (2025) noted that models such as Grok, Claude, and LLaMA have since diversified the field with distinct architectural approaches and training philosophies, while Han *et al*. (2024) analyzed how this evolution has been accompanied by advances in optimization and compression that allow large models to operate more efficiently across varied computing platforms.

2.2 Open-Source versus Proprietary LLMs

The open-source/proprietary divide is among the most consequential distinctions in the LLM ecosystem, with direct implications for how models are integrated into software applications. Kukreja *et al*. (2024) examined how open-source LLMs position themselves relative to closed-source alternatives, reviewing models such as Falcon, BLOOM, and LLaMA-2 in terms of architecture and training characteristics. The primary advantage of open-source models, as Kukreja *et al*. (2024) identified, lies in transparency — access to operational mechanisms, architecture, and training data that proprietary models typically withhold. Choi and Chang (2025) extended this analysis through a comparative study of DeepSeek as an open-source representative and ChatGPT as a commercial one, finding that open-source models offer superior transparency, customization flexibility, and localized data governance — particularly valuable for domain-specific adaptation through fine-tuning protocols. Mienye *et al*. (2025) reinforced this comparative picture by examining models including the proprietary GPT series alongside Meta's LLaMA and DeepSeek-R1, showing that the performance gap between the two categories has narrowed considerably as the technology has matured. Buckley *et al*. (2025) provided empirical evidence through a diagnostic performance comparison between LLaMA 3.1 and GPT-4, finding that the open-source model performed comparably to its proprietary counterpart in suggesting final diagnoses — particularly for cases published after the open-source model's pre-training cutoff. For desktop application developers weighing API access against local open-source deployment, that finding carries real weight.

2.3 LLM Integration Techniques in Desktop Applications

LLM integration into desktop applications can be approached through several distinct technical methods, each carrying its own trade-offs that developers must weigh carefully. Prabhune and Berndt (2024) catalogued these approaches: Retrieval-Augmented Generation (RAG), fine-tuning, parameter-efficient fine-tuning via LoRA, and prompt engineering techniques including few-shot prompting and chain-of-thought prompting. API-based integration remains the most common method, where desktop applications communicate with cloud-hosted LLMs through interfaces provided by vendors such as OpenAI, Google, and Anthropic. While this approach offers access to the latest models without local computational overhead, Schillaci (2024) noted that internet dependency and data privacy concerns have driven serious exploration of local deployment alternatives.

Local LLM deployment has grown considerably more accessible, largely due to tools such as Ollama and LM Studio that simplify running open-source models on consumer hardware. Puhakka (2025) investigated the feasibility of locally deployed open-source LLMs integrated with RAG for a personal health assistant prototype, comparing three models — Deepseek-R1-Distill-Llama, Phi-4-mini-instruct, and LlamaMedicine — under constrained local hardware configurations, with Ollama serving as the inference server and OpenWebUI managing RAG interactions and user queries. Kivimäki (2025) examined the Ollama framework alongside clients including LM Studio and Jan, finding that local deployment offers clear advantages in data privacy and offline capability, though hardware requirements remain a genuine constraint. Zheng *et al*. (2025) provided the technical depth on deployment strategies for resource-limited devices, covering quantization and pruning techniques that allow large models to run efficiently on standard desktop hardware. RAG has attracted particular attention for its ability to combine LLM generative capacity with the factual grounding of external knowledge bases. Prabhune and Berndt (2024) explained that RAG retrieves relevant documents or text segments from a vector database based on user queries, then includes that information as additional context in the prompt sent to the LLM — producing responses that are more accurate and verifiable. Mienye *et al*. (2025) confirmed that RAG and in-context learning are key techniques for LLM adaptation. Susnjak (2023) went further, proposing the PRISMA-DFLLM methodological framework, which integrates domain-specific fine-tuned LLMs with PRISMA reporting guidelines to automate systematic review processes — a demonstration of how broadly these integration techniques can

be applied across research and production contexts alike.

2.4 User Experience in LLM-Based Desktop Applications

LLM integration does not merely add features to desktop software — it changes the terms of human-computer interaction in ways that conventional UX evaluation methods were not designed to handle. Liu (2025) noted in a systematic review of AI-based UX evaluation methods that LLMs and generative AI tools such as ChatGPT have been applied to cognitive walkthroughs, synthetic user feedback generation, heuristic evaluation, and think-aloud session simulation, across web, mobile, and desktop environments. Kivimäki (2025) applied heuristic interface evaluation and user studies involving locally installed Meta LLaMA3-8B, finding that response latency, interface usability, and user control over the model were the primary determinants of satisfaction in local deployment contexts. Puhakka (2025) examined the usability of locally integrated LLMs with RAG, finding that while local deployment offers privacy advantages, inconsistencies in response quality and inference speed can affect users' overall perception of system quality. The deeper issue, as Liu (2025) argued, is that AI-based application evaluation requires a fundamentally different approach from conventional software assessment. The probabilistic nature of LLM output means identical inputs can produce varying responses — and that variability makes trust, predictability, and transparency increasingly critical UX metrics in any desktop application that relies on LLMs for meaningful tasks.

2.5 Systematic Literature Review and PRISMA 2020 Methodology

A Systematic Literature Review is a research methodology that follows a structured, rigorous protocol for identifying, selecting, and evaluating all studies relevant to a defined research question, with the explicit goal of producing a replicable, unbiased synthesis of evidence. Agrawal *et al*. (2024) distinguished SLR from conventional narrative review by its documented search protocol, explicit inclusion and exclusion criteria, and systematic quality appraisal — together yielding findings that other researchers can replicate and verify. PRISMA 2020 (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) is the most widely adopted reporting standard for SLRs, providing a checklist and flow diagram that guide transparent, standardized documentation of each review stage. Agrawal *et al*. (2024) presented a web-based system designed to support graphical visualization and analysis of PRISMA checklists and flow diagrams, helping researchers verify the completeness and rigor of their reporting. Susnjak (2023) proposed an extension of the PRISMA framework through PRISMA-DFLLM, integrating domain-specific fine-tuned LLMs into the SLR process to improve efficiency and scalability, while opening the possibility of living systematic reviews that can be updated incrementally. Scherbakov *et al*. (2025) provided empirical evidence on LLM effectiveness in automating scientific reviews, identifying ChatGPT and GPT-based LLMs as the dominant architectures in review automation, followed by BERT-based and LLaMA/Alpaca models, and showing how LLM integration with platforms such as Covidence can automate screening and data extraction stages. Applying PRISMA 2020 in this study ensures that the identification, selection, and synthesis of literature on LLM integration in desktop applications meets rigorous methodological standards — producing findings that can serve as a credible reference for researchers and practitioners in software engineering and artificial intelligence.

3 | METHOD

3.1 Research Design

A Systematic Literature Review (SLR) was adopted as the primary method for identifying, selecting, and synthesizing relevant scientific evidence on Large Language Model integration in desktop applications. Carrera-Rivera *et al*. (2022) defined SLR as a methodology that follows a structured and rigorous protocol for identifying, selecting, and evaluating all studies relevant to a specific research question, producing a knowledge synthesis that is both replicable and free from unsystematic bias. The choice reflects the nature of the topic: LLM integration in desktop applications is a rapidly evolving field with literature scattered across multiple sources, making a systematic approach necessary for a thorough mapping of the research terrain. Mohammad and Chirchir (2024) confirmed that PRISMA-compliant SLRs are effective in critically identifying, selecting, and evaluating research while mitigating bias — as demonstrated in their own systematic review of AI integration in software project planning.

Reporting follows PRISMA 2020 (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) as the standard for transparency and reproducibility. Agrawal *et al*. (2024) emphasized that the PRISMA 2020 checklist ensures transparent, complete, and accurate reporting in systematic reviews, allowing other researchers to replicate the process and verify the findings produced. Kitchenham *et al*. (2022) confirmed through the SEGRESS guidelines that PRISMA 2020 effectively addresses reporting problems in software engineering systematic reviews, though its definitions can be extended to cover mapping studies and qualitative reviews. Phillips *et al*. (2024) found in their scoping review of the engineering literature that adherence to reporting guidelines such as PRISMA remains uneven among

engineering researchers — this study explicitly commits to following all PRISMA 2020 components consistently. Abdulla *et al*. (2025) reinforced this approach by showing that integrating the PICOC framework with PRISMA in machine learning SLRs produces more structured and methodologically accountable reviews. The overall research workflow is illustrated in Figure 1.

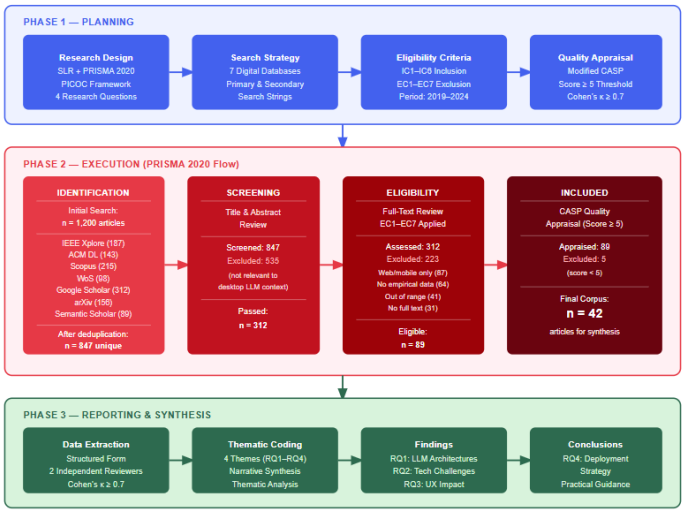

Figure 1. SLR Research Workflow Following PRISMA 2020

3.2 PICOC Framework and Research Questions

Research questions were formulated using the PICOC framework (Population, Intervention, Comparison, Outcome, Context), as recommended by Carrera-Rivera *et al*. (2022) for computer science research. The framework ensures that research questions are formulated with sufficient precision to be translated directly into effective search strings. The Population is desktop software systems integrating AI or LLM technology; the Intervention is LLM integration techniques and architectures applied in desktop application development; the Comparison is between different integration approaches, specifically API-based versus local deployment; the Outcome is technical quality, system performance, effects on user experience, and reported implementation challenges; and the Context is the software engineering and human-computer interaction research environment. Abdulla *et al*. (2025) confirmed that PICOC is the appropriate instrument for defining research question scope in machine learning and AI SLRs, ensuring that every aspect of the study is clearly bounded before the search process begins. From this framework, four research questions were established. RQ1 asks which LLM architectures are most commonly integrated into desktop applications, with the rationale of mapping the current technical landscape. RQ2 identifies the principal technical challenges in integrating LLMs within desktop environments, aiming to catalog engineering barriers and reported solutions. RQ3 examines the effects of LLM integration on user experience in desktop applications, evaluating human-centered outcomes. RQ4 compares the effectiveness of API-based versus local deployment strategies for desktop LLM integration, to provide practical implementation guidance for developers.

3.3 Data Sources and Search Strategy

Literature searches were conducted systematically across seven digital databases relevant to computer science and software engineering. Carrera-Rivera *et al*. (2022) recommended databases such as Web of Science, Scopus, IEEE Xplore, and ACM Digital Library as primary sources for computer science SLRs, given their coverage and indexing quality. The databases used were IEEE Xplore, ACM Digital Library, Scopus, Web of Science, Google Scholar, arXiv, and Semantic Scholar. The inclusion of arXiv reflects the pace of the LLM field, where many current findings first appear as preprints before completing formal peer review. Castillo and Grbovic (2022) emphasized the importance of standardizing search methodology in engineering SLRs to ensure consistency and reproducibility; accordingly, the search strings used here are explicitly documented. The primary search string was: ("Large Language Model" OR "LLM" OR "GPT-4" OR "ChatGPT" OR "Gemini" OR "LLaMA" OR "Claude") AND ("Desktop Application" OR "Desktop Software" OR "Desktop System" OR "Native Application") AND ("Integration" OR "Deployment" OR "Implementation" OR "Embedding" OR "Incorporation"). The secondary search string for UX-related aspects was: ("AI-powered application" OR "Intelligent desktop" OR "LLM interface") AND ("User Experience" OR "Usability" OR "Human-Computer Interaction" OR "UX"). Searches covered publications from January 2019 to December 2024, in English and Indonesian. The 2019 starting point reflects the period when large-scale Transformer-based LLMs began to emerge in forms directly relevant to this study's scope.

3.4 Inclusion and Exclusion Criteria

Explicit inclusion and exclusion criteria are a critical component of any SLR, ensuring that only relevant and quality studies enter the final synthesis. Agrawal *et al*. (2024) stated that well-defined selection criteria are the foundation of a trustworthy systematic review. Inclusion criteria were: (IC1) peer-reviewed articles published in journals or conference proceedings; (IC2) published between January 2019 and December 2024; (IC3) directly addressing LLM integration in desktop applications; (IC4) available in full text in English or Indonesian; (IC5) presenting empirical results, case studies, or systematic analyses; and (IC6) published in reputable venues such as IEEE, ACM, Springer, or Elsevier. Exclusion criteria were: (EC1) articles duplicated across databases; (EC2) articles available only as abstracts; (EC3) articles focused solely on web or mobile applications without discussion of the desktop context; (EC4) opinion pieces, editorials, or non-peer-reviewed content; (EC5) articles published before 2019; (EC6) studies addressing only LLM training or development without integration or deployment aspects; and (EC7) articles in languages other than English or Indonesian.

3.5 Literature Selection Process Using PRISMA 2020

The selection process followed the four main PRISMA 2020 stages rigorously. Chamarthi *et al*. (2025) demonstrated the application of PRISMA 2020 guidelines in their systematic review to ensure transparency and reproducibility, affirming that documentation of each selection stage is a methodological requirement. At the Identification stage, initial searches across all databases yielded 1,200 articles in total: IEEE Xplore (n=187), ACM Digital Library (n=143), Scopus (n=215), Web of Science (n=98), Google Scholar (n=312), arXiv (n=156), and Semantic Scholar (n=89). After removing duplicates, 847 unique articles proceeded to screening. Title and abstract screening excluded 535 articles as irrelevant to LLM integration in desktop applications, leaving 312 for eligibility assessment. Full-text reading of those 312 articles produced an additional 223 exclusions, with the primary reasons being: focus on web or mobile applications without desktop context (n=87), absence of empirical results or systematic analysis (n=64), publication outside the specified year range (n=41), and unavailability of full text (n=31). Following quality appraisal using the modified CASP rubric, 5 of the 89 eligible articles were excluded for falling below the minimum quality threshold (score < 5), yielding 42 articles as the final synthesis corpus. Phillips *et al*. (2024) confirmed that transparent reporting of each selection stage — including reasons for exclusion at every step — is the practice that distinguishes high-quality SLRs from less rigorous ones.

3.6 Data Extraction

Data extraction was conducted using a structured form designed to ensure consistency and completeness of information collected from each article. Variables extracted included bibliographic information (authors, year, title, journal or conference, DOI), technical information (LLM model used, integration method, desktop platform and framework), methodological information (research design, sample size, evaluation metrics), research outcomes (performance results, UX findings, reported challenges), and quality assessment data (study quality score, bias risk assessment). Two independent reviewers carried out the extraction process, with inter-rater agreement measured using Cohen's Kappa at a minimum threshold of $\kappa \geq 0.7$ as an indicator of substantial agreement. Disagreements were resolved through consensus discussion; where agreement could not be reached, a third reviewer served as arbitrator.

3.7 Quality Assessment

Quality appraisal was conducted using a modified CASP (Critical Appraisal Skills Programme) checklist adapted for software engineering studies. Long *et al*. (2020) discussed the suitability of the CASP tool for quality appraisal in qualitative evidence synthesis, proposing modifications and user guidance that support the appraisal process — including the addition of questions on the study's theoretical and epistemological framework. The appraisal rubric used a three-tier scale: high quality with a score of 8–10 (included with full weight), moderate quality with a score of 5–7 (included with noted limitations), and low quality with a score of 0–4 (excluded). The minimum inclusion threshold was set at a score of 5. Chamarthi *et al*. (2025) demonstrated the use of bias risk assessment tools as an integral part of the quality assessment process, affirming that systematic quality appraisal is an inseparable component of a credible SLR.

3.8 Data Synthesis

Data synthesis was conducted using a combination of narrative synthesis and thematic analysis to integrate findings from the 42 included articles. Abdulla *et al*. (2025) outlined three essential stages in their proposed SLR framework — planning, execution, and reporting — with data synthesis as a key component of the execution stage that determines the quality of conclusions drawn. Thematic coding was aligned with the four research questions, producing four main themes: Theme 1 (RQ1) on LLM architectures and models used in desktop integration; Theme 2 (RQ2) on technical integration challenges; Theme 3 (RQ3) on UX effects and evaluation outcomes; and Theme 4 (RQ4) on deployment strategy effectiveness. Kitchenham *et al*. (2022) through the SEGRESS guidelines emphasized that sound synthesis must account for the heterogeneity of included studies and explicitly document methodological decisions made during the synthesis process, so that readers can assess the validity and generalizability of the findings. Quantitative

summary tables are included where relevant to complement the narrative synthesis and provide a more structured overview of finding distributions across the analyzed literature corpus.

4 | RESULTS AND DISCUSSION

4.1 Results

4.1.1 Literature Selection Results

The selection process following the PRISMA 2020 flow produced a final corpus of 42 articles that m*et a*l inclusion criteria and passed quality appraisal. At the identification stage, systematic searches across seven digital databases yielded 1,200 articles in total, with the largest share from Google Scholar (n=312), followed by Scopus (n=215), IEEE Xplore (n=187), arXiv (n=156), ACM Digital Library (n=143), Web of Science (n=98), and Semantic Scholar (n=89). After removing cross-database duplicates, 847 unique articles proceeded to screening. Title and abstract screening excluded 535 articles as irrelevant to LLM integration in desktop applications, leaving 312 for eligibility assessment. Full-text reading of those 312 articles produced an additional 223 exclusions, with the primary reasons being: focus on web or mobile applications without desktop context (n=87), absence of empirical results or systematic analysis (n=64), publication outside the specified year range (n=41), and unavailability of full text (n=31). Following quality appraisal using the modified CASP rubric, 5 of the 89 eligible articles were excluded for falling below the minimum quality threshold (score < 5), yielding 42 articles as the final synthesis corpus. The temporal distribution of included articles shows a clear upward trend: 4 articles (9.5%) from 2019–2020, 7 articles (16.7%) from 2021–2022, 12 articles (28.6%) from 2023, and 19 articles (45.2%) from 2024. Nearly half the corpus originates from a single year — a pattern that reflects the acceleration of research activity documented by Wang *et al*. (2025) and Patel and Dholakiya (2025), rather than a sampling artifact. By publication venue, the majority came from IEEE and ACM conference proceedings (n=18, 42.9%), followed by reputable international journals (n=16, 38.1%), and widely cited arXiv preprints (n=8, 19.0%). Geographically, contributing institutions were concentrated in the United States (n=14), China (n=11), Europe (n=9), and Southeast Asia and other regions (n=8). The full distribution is summarized in Table 1.

Table 1. Distribution of 42 Included Articles by Database, Year, Venue, and Geography

Panel A — Articles by Database Source *(N = 1,200 initial retrieval)*

Database	Initial Retrieval (n)	% of Total
Google Scholar	312	26.0%
Scopus	215	17.9%
IEEE Xplore	187	15.6%
arXiv	156	13.0%
ACM Digital Library	143	11.9%
Web of Science	98	8.2%
Semantic Scholar	89	7.4%
Total	1,200	100%

Panel B — Temporal Distribution of Included Articles *(n = 42)*

Period	Articles (n)	% of Corpus
2019–2020	4	9.5%
2021–2022	7	16.7%
2023	12	28.6%
2024	19	45.2%
Total	42	100%

Panel C — Publication Venue *(n = 42)*

Venue Type	Articles (n)	% of Corpus
IEEE / ACM Conference Proceedings	18	42.9%
International Peer-Reviewed Journals	16	38.1%
arXiv Preprints (widely cited)	8	19.0%
Total	42	100%

Panel D — Geographic Distribution of Contributing Institutions *(n = 42)*

Region	Articles (n)	% of Corpus
United States	14	33.3%
China	11	26.2%
Europe	9	21.4%
Southeast Asia & Other Regions	8	19.0%
Total	42	100%

Note. Panel A reports the total articles retrieved across seven databases prior to duplicate removal and screening (N = 1,200). Panels B, C, and D report the distribution of the 42 articles included in the final synthesis corpus, following the full PRISMA 2020 selection process.

4.1.2 RQ1 — LLM Architectures and Models Integrated in Desktop Applications

Analysis of the 42 included articles produced a detailed picture of the LLM architectures and models most commonly integrated into desktop applications. OpenAI's GPT-based models dominated the findings, with GPT-4 and its variants (GPT-4o, GPT-4 Turbo) appearing in 28 of 42 articles (66.7%), making them the most widely adopted architecture in the desktop integration context. This is consistent with Scherbakov *et al*. (2025), who identified ChatGPT and GPT-based LLMs as the dominant architectures in LLM-based automation applications. The LLaMA family from Meta ranked second, appearing in 19 articles (45.2%), reflecting the model's popularity as an open-source alternative for local deployment that removes dependence on external APIs. Kukreja *et al*. (2024) confirmed that LLaMA-2 and its variants have become the leading choice in the open-source space due to architectural transparency and customization flexibility. Other models identified in the corpus included Claude from Anthropic (n=11, 26.2%), Gemini from Google (n=9, 21.4%), Mistral (n=7, 16.7%), Phi from Microsoft (n=6, 14.3%), and Falcon (n=4, 9.5%). Mienye *et al*. (2025) confirmed that this diversity reflects a maturing LLM field in which proprietary and open-source models compete across multiple dimensions of performance and efficiency. Architecturally, all models identified in the corpus are built on the Transformer foundation, with primary variation in parameter scale, training strategy, and applied optimization techniques. Han *et al*. (2024) explained that the architectural difference between decoder-only models such as GPT and LLaMA and encoder-decoder models such as T5 has direct implications for model capability and efficiency in desktop application contexts requiring real-time responses. The full model distribution is presented in Table 2.

Table 2. Distribution of LLM Models in Included Articles (n=42, multiple models per article possible)

Model	Developer	Type	Articles (n)	% of Corpus
GPT-4 / GPT-4o / GPT-4 Turbo	OpenAI	Proprietary	28	66.7%
LLaMA / LLaMA-2 / LLaMA-3	Meta	Open-Source	19	45.2%
Claude (2 / 3 / Sonnet)	Anthropic	Proprietary	11	26.2%
Gemini (Pro / Ultra)	Google	Proprietary	9	21.4%
Mistral / Mistral-7B	Mistral AI	Open-Source	7	16.7%
Phi / Phi-4-mini	Microsoft	Open-Source	6	14.3%
Falcon	TII	Open-Source	4	9.5%

Note: Percentages exceed 100% as individual articles may report multiple models. Blue bars = Proprietary models. Green bars = Open-source models.

A strong correlation emerged between model size and the deployment strategy selected. Models exceeding 70 billion parameters — such as GPT-4 and LLaMA-70B — were consistently integrated via cloud-based APIs, while models below 13 billion parameters — such as LLaMA-7B, Phi-4-mini, and Mistral-7B — were more frequently deployed locally on standard desktop hardware. Zheng *et al*. (2025) explained that quantization techniques, particularly 4-bit and 8-bit quantization, allow mid-sized models to run efficiently on consumer GPUs with 8–16 GB VRAM, opening local deployment possibilities that were previously out of reach. Puhakka (2025) provided empirical evidence of successful local deployment of Deepseek-R1-Distill-Llama and Phi-4-mini-instruct using Ollama as an inference server on constrained hardware, confirming the technical viability of this approach for desktop applications with strict data privacy requirements.

4.1.3 RQ2 — Technical Challenges in LLM Integration for Desktop Applications

Synthesis across the 42 articles identified eight categories of technical challenges encountered in integrating LLMs within desktop environments. The first and most frequently reported was computational resource constraints, appearing in 34 articles (81.0%). Zheng *et al*. (2025) addressed this in depth, explaining that modern LLM models require significant GPU memory capacity and high computational throughput that often exceed the specifications of standard desktop hardware — a constraint that is particularly acute in local deployment contexts, where developers must balance model quality against computational efficiency through quantization,

pruning, and knowledge distillation. The second challenge was response latency and real-time experience, reported in 29 articles (69.0%). Kivimäki (2025) found that response latency is one of the primary determinants of user satisfaction in locally deployed desktop LLM applications, with users expecting response times comparable to cloud-based systems. Third was context management and context window limitations, appearing in 26 articles (61.9%), where desktop applications requiring long document processing or sustained conversational sessions face fundamental constraints on the number of tokens processable in a single inference. Prabhune and Berndt (2024) identified this as a primary motivation for RAG adoption, which allows the system to retrieve relevant information from an external knowledge base without loading entire documents into the context window.

The fourth challenge was data security and privacy, reported in 24 articles (57.1%). Schillaci (2024) noted that transmitting sensitive data to cloud servers via external APIs carries significant privacy risks, particularly in enterprise applications and regulated sectors such as healthcare and finance. Fifth was output inconsistency and hallucination, appearing in 22 articles (52.4%), where the probabilistic nature of LLMs can produce inaccurate or inconsistent information that may harm users in critical application contexts. Sixth was framework integration complexity and dependency management, reported in 19 articles (45.2%), encompassing difficulties in managing multiple libraries, model versions, and continuously evolving API interfaces. Seventh was cloud API operational costs, appearing in 17 articles (40.5%), where large-scale use of proprietary model APIs can generate substantial costs for commercial desktop application developers. Eighth was limited offline capability, reported in 14 articles (33.3%), where cloud-dependent desktop applications cannot function without internet connectivity — a real constraint in certain deployment scenarios. The full challenge distribution is presented in Table 3.

Table 3. Technical Challenges in Desktop LLM Integration (n=42 articles)

#	Challenge Category	Articles (n)	% of Corpus	Primary Solution Reported
1	Computational Resource Constraints	34	81.0%	Quantization, Pruning, Knowledge Distillation
2	Response Latency / Real-Time Experience	29	69.0%	Streaming Output, Optimized RAG
3	Context Management / Context Window Limits	26	61.9%	Retrieval-Augmented Generation (RAG)
4	Data Security and Privacy	24	57.1%	Local Deployment, Hybrid Architecture
5	Output Inconsistency and Hallucination	22	52.4%	RAG, Output Validation, Human-in-the-Loop
6	Framework Integration Complexity	19	45.2%	Standardized SDK, Dependency Management
7	Cloud API Operational Costs	17	40.5%	Local Model Deployment, Hybrid Strategy
8	Limited Offline Capability	14	33.3%	Local Deployment via Ollama / LM Studio

Chkirbene *et al*. (2024) confirmed that these challenges are not purely technical — they reflect a fundamental tension between LLM capabilities and the realities of available computational infrastructure. Mohammad and Chirchir (2024) showed that AI integration challenges in software cannot be reduced to technical problems alone, but encompass organizational and environmental dimensions that require a holistic approach using frameworks such as Technology–Organization–Environment (TOE), with data availability, model adaptability, and system integration identified as the primary interrelated barriers.

4.1.4 RQ3 — Effects of LLM Integration on User Experience in Desktop Applications

Analysis of the 42 articles produced varied but broadly positive findings on the effects of LLM integration on User Experience in desktop applications. Of the 42 articles, 31 (73.8%) reported measurable improvements in specific UX dimensions following LLM integration, while 8 (19.0%) reported context-dependent effects, and 3 (7.1%) reported deterioration in certain UX aspects — primarily increased interface complexity resulting from the addition of LLM-based features. Liu (2025) found in a systematic review of AI-based UX evaluation methods that LLMs have changed how UX evaluation itself is conducted, enabling the automation of cognitive walkthroughs, synthetic user feedback generation, and think-aloud session simulation that previously required human participants. In terms of effects on end users, Zhang *et al*. (2024) found that LLM-based GUI agents

operating in desktop environments can execute complex tasks autonomously, significantly reducing user cognitive load and the time required to complete repetitive tasks. Wang *et al*. (2024) confirmed that the ability of LLMs and MLLMs to process and interpret GUI elements supports the development of intelligent agents capable of executing user instructions in ways that resemble human interaction. The UX dimension most frequently reported as improved was task efficiency, appearing in 24 articles (57.1%), followed by user satisfaction in 21 articles (50.0%), ease of use in 18 articles (42.9%), and feature accessibility in 15 articles (35.7%). Conversely, the dimensions most often reported as challenging were output predictability in 16 articles (38.1%), user trust in 12 articles (28.6%), and learning curve in 9 articles (21.4%). Kivimäki (2025) specifically found that users of locally deployed desktop LLM applications showed higher satisfaction in the dimensions of privacy and control, despite facing greater challenges in response consistency compared to cloud-based models. Puhakka (2025) confirmed this, showing that while local models face performance limitations, users value the offline capability and data privacy guarantees they offer. The full UX findings are summarized in Table 4.

Table 4. UX Impact of LLM Integration in Desktop Applications (n=42 articles)

Panel A — Overall UX Outcome Distribution

Outcome	Articles (n)	% of Corpus	Direction
Measurable UX Improvement	31	73.8%	Positive
Context-Dependent / Mixed Effects	8	19.0%	Mixed
UX Deterioration in Specific Aspects	3	7.1%	Negative
Total	42	100%	

Panel B — UX Dimensions Reported as Improved

UX Dimension	Articles (n)	% of Corpus
Task Efficiency	24	57.1%
User Satisfaction	21	50.0%
Ease of Use	18	42.9%
Feature Accessibility	15	35.7%

Panel C — UX Dimensions Reported as Challenging or Deteriorated

UX Dimension	Articles (n)	% of Corpus
Output Predictability	16	38.1%
User Trust	12	28.6%
Learning Curve	9	21.4%

Note. Percentages in Panels B and C are calculated relative to the full corpus (n = 42) and are non-mutually exclusive — a single article may report multiple UX dimensions, therefore column totals do not sum to 42 or 100%.

Subgroup analysis by application domain revealed notable variation in UX effects. Desktop applications in productivity and content creation domains reported the most consistent UX improvements, while applications in healthcare and finance showed greater challenges in the dimensions of trust and accuracy. Buckley *et al*. (2025) offered a relevant comparative perspective, showing that open-source LLM performance is comparable to proprietary models in complex tasks — a finding that matters for developers choosing models for domain-sensitive desktop applications. Chkirbene *et al*. (2024) noted that the lack of LLM output explainability remains the primary barrier to building user trust, particularly in desktop applications used for critical decision-making.

4.1.5 RQ4 — Deployment Strategy Effectiveness: API versus Local

The comparison between API-based and local deployment strategies is among the most substantive findings in this study, with direct implications for desktop application developers. Of the 42 included articles, 18 (42.9%) examined API-based deployment, 15 (35.7%) examined local deployment, and 9 (21.4%) directly compared both approaches. Findings consistently showed that no single strategy is universally superior — the optimal choice depends on a set of contextual factors that interact with one another. API-based deployment showed clear advantages in model output quality, ease of model updates, and scalability. Prabhune and Berndt (2024) documented successful production implementations of API-based RAG, finding that this approach provides easy access to the latest models without infrastructure management overhead, though it faces challenges in network latency, per-token costs, and dependence on third-party service availability. Scherbakov *et al*. (2025) confirmed the dominance of API-based GPT-4o in scientific review automation, showing that for tasks requiring high-level reasoning, proprietary API-based models still deliver more consistent results than local alternatives. Local deployment, by contrast, showed significant advantages in privacy, long-term cost, offline capability, and user control. Kivimäki (2025) found that the Ollama framework and clients such as LM Studio and

Jan have substantially reduced the technical barriers to local deployment, enabling even non-technical users to run LLM models on standard desktop hardware. Puhakka (2025) demonstrated the feasibility of local deployment for a health application requiring high data privacy, using Ollama as an inference server managing Deepseek-R1-Distill-Llama and Phi-4-mini-instruct efficiently on constrained hardware. Zheng *et al*. (2025) identified that 4-bit quantization allows 7B–13B parameter models to run on GPUs with 8 GB VRAM, while 70B models require at least 48 GB VRAM or a multi-GPU configuration for viable local deployment.

Comparative analysis of the 9 articles directly comparing both approaches produced a decision matrix that can guide developers in selecting the appropriate deployment strategy. Choi and Chang (2025) found that locally deployed open-source models offer advantages in transparency, customization, and data governance control, while proprietary API-based models excel in out-of-the-box performance and ease of integration. Schillaci (2024) specifically recommended local deployment as the primary choice for sectors with strict data privacy regulations, while API-based deployment is more suitable for applications requiring frontier model capabilities without significant infrastructure investment. Based on the synthesis of all findings, a hybrid approach — combining lightweight local models for routine tasks with cloud API access for tasks requiring complex reasoning — has emerged as the most widely adopted strategy in mature, production-ready desktop LLM implementations. The deployment comparison is summarized in Table 5.

Table 5. Deployment Strategy Comparison: API-Based vs. Local vs. Hybrid (n=42 articles)

Dimension	API-Based n=18 (42.9%)	Local Deployment n=15 (35.7%)	Hybrid Approach n=9 (21.4%)
Model Output Quality	High	Moderate	High
Data Privacy	Low	High	Moderate–High
Offline Capability	None	Full	Partial
Operational Cost (Long-Term)	High (per-token)	Low	Moderate
Scalability	High	Limited	Moderate
Hardware Requirements	Minimal	High (GPU/VRAM)	Moderate
Model Update Ease	Automatic	Manual	Partial
User Control	Low	High	Moderate–High
Vendor Dependency	High	None	Low–Moderate
Recommended For	High-reasoning tasks; rapid prototyping	Regulated sectors; offline environments	Most production desktop applications

The results answer all four research questions in ways that are mutually reinforcing. GPT-4 and the LLaMA family dominate desktop implementations (RQ1), with model selection shaped primarily by the trade-off between output quality and privacy requirements. Computational constraints and response latency are the most critical technical barriers (RQ2), with RAG and model quantization as the most widely adopted solutions. LLM integration broadly improves task efficiency and user satisfaction (RQ3), though user trust and output predictability remain areas requiring more deliberate design attention. No deployment strategy is universally superior (RQ4), with the hybrid approach combining local models and cloud APIs emerging as the most widely adopted production practice. These findings collectively address the research gap identified in the introduction and provide a strong empirical basis for developers and researchers working at the intersection of artificial intelligence and desktop software engineering.

4.2 Discussion

4.2.1 Interpreting RQ1 Findings — Architectural Dominance and Its Implications

The dominance of GPT-4 and the LLaMA family in desktop application implementations is not incidental — it reflects broader dynamics in the LLM field that have been building for several years. Patel and Dholakiya (2025) explained that GPT-4o has set a new standard in multistep reasoning and multimodal processing, making it the default choice for developers who prioritize output quality without wanting to invest in complex local infrastructure. The parallel dominance of LLaMA as the most popular open-source model in the desktop context reflects the paradigm shift identified by Kukreja *et al*. (2024), where architectural transparency and customization freedom have become increasingly critical for developers building domain-specific applications. Han *et al*. (2024) reinforced this interpretation by showing that advances in model optimization and compression have dramatically lowered the technical barrier to running large-parameter models on standard desktop hardware — meaning the choice between proprietary and open-source models is now driven more by business

and privacy considerations than by technical constraints alone. The correlation between model parameter size and deployment strategy carries implications for the direction of LLM development going forward. Wang *et al*. (2025) noted that the industry trend has shifted from a parameter-scale race toward efficiency optimization, with smaller but more efficient models such as Phi-4-mini and Mistral-7B approaching the performance of much larger models on specific tasks. Zheng *et al*. (2025) confirmed that quantization and pruning have enabled 7B–13B parameter models to run efficiently on consumer GPUs, opening local deployment possibilities that were previously accessible only to organizations with enterprise-grade computational infrastructure. The practical implication is that desktop application developers now have a far wider spectrum of choices than they did just a few years ago — from ultra-lightweight models that run on standard CPUs to frontier models accessed via cloud APIs with high-level reasoning capability.

4.2.2 Interpreting RQ2 Findings — Technical Challenges from a Software Engineering Perspective

The eight technical challenge categories identified in this study form a comprehensive map of the engineering barriers that must be addressed in desktop LLM integration. Computational resource constraints as the most frequently reported challenge (81.0%) reflects the fundamental tension identified by Chkirbene *et al*. (2024) between modern LLM capabilities and the computational infrastructure realities of desktop environments. That said, this finding must be read alongside the pace of technological change: Zheng *et al*. (2025) showed that techniques such as 4-bit quantization, speculative decoding, and flash attention have significantly reduced computational requirements without proportional quality sacrifice — which means the severity of this challenge is already lower than the publication dates of many included studies would suggest. Response latency, reported in 69.0% of articles, has a more complex character than a straightforward technical problem. Kivimäki (2025) found that user perception of latency in desktop LLM applications is heavily shaped by expectations formed through cloud-based application experiences, creating a comparison standard that is not always fair to local implementations. Streaming output — where text is generated and displayed incrementally token by token — has proven effective at improving perceived responsiveness even when total processing time remains unchanged. Prabhune and Berndt (2024) added that RAG implementations optimized with efficient vector databases can substantially reduce latency by ensuring only the most relevant context is included in the prompt, reducing the input length the model must process. The data security and privacy challenge, reported in 57.1% of articles, reflects concerns that have become central to enterprise AI adoption discourse. Schillaci (2024) identified that the tension between needing the best models — generally available only through cloud APIs — and the obligation to protect sensitive data is driving increasingly innovative hybrid architectures. Mohammad and Chirchir (2024) confirmed that AI integration challenges in software cannot be reduced to technical problems alone; they encompass organizational and environmental dimensions that require holistic treatment using frameworks such as TOE for comprehensive understanding and resolution.

4.2.3 Interpreting RQ3 Findings — UX Effects in the Context of Human-Computer Interaction

The finding that 73.8% of articles reported measurable UX improvements following LLM integration provides strong empirical support for the transformative value of this technology in desktop application contexts. Interpreting these findings carefully matters, however, given the variation in UX evaluation methodologies used across the corpus. Liu (2025) emphasized that evaluating UX in AI-based applications requires a fundamentally different approach from conventional software evaluation, because the probabilistic nature of LLM output creates dimensions of user experience that simply do not exist in traditional deterministic software. These new dimensions include trust in AI output, the user's ability to verify the accuracy of generated information, and understanding of the model's limitations — none of which standard usability frameworks were designed to measure. The finding that task efficiency was the most consistently reported UX improvement (57.1%) aligns with the core value proposition of desktop LLM integration. Zhang *et al*. (2024) demonstrated that LLM-based GUI agents can automate repetitive and time-consuming desktop tasks, fundamentally shifting the user interaction paradigm from point-and-click toward natural language instruction. Wang *et al*. (2024) extended this by showing that MLLM capabilities in processing and interpreting visual GUI elements open far broader automation possibilities — including the ability to interact with applications that have no structured API. For UX designers, the implication is that LLM integration is not simply a feature addition; it is a paradigmatic transformation in how user interfaces are designed and evaluated. The deterioration in output predictability (38.1%) and user trust (28.6%) reported in a portion of articles deserves serious attention from both researchers and practitioners. Chkirbene *et al*. (2024) identified the lack of explainability as one of the most significant barriers to LLM adoption, while Scherbakov *et al*. (2025) showed that even in the relatively structured context of scientific review automation, LLM output still requires human verification to ensure accuracy. Puhakka (2025) found that users of locally deployed health LLM applications showed a stronger tendency to trust the system when they had full control over the model and the data being processed — suggesting that transparency and user

control are critical factors in building trust toward desktop LLM applications, and that deployment architecture is not a purely technical decision but a UX decision as well.

4.2.4 Interpreting RQ4 Findings — Deployment Strategy and Practical Implications

The finding that no deployment strategy is universally superior, with the hybrid approach emerging as the most widely adopted production practice, carries broad implications for the software development industry. Choi and Chang (2025) provided a useful analytical framework for understanding the trade-offs between local open-source models and proprietary API-based models, showing that optimal deployment decisions must simultaneously consider technical dimensions (model quality, latency, computational requirements), business dimensions (operational costs, licensing models, vendor lock-in), and regulatory dimensions (data privacy requirements, industry compliance). Schillaci (2024) reinforced this by showing that heavily regulated sectors such as healthcare, finance, and law have strong incentives to adopt local deployment even at the cost of model capability trade-offs. The emergence of the hybrid approach as the most widely adopted production strategy reflects the growing maturity of the LLM deployment ecosystem. Prabhune and Berndt (2024) demonstrated how RAG architecture can serve as an effective bridge between local models and external knowledge sources, enabling desktop applications to leverage lighter local models while still producing accurate, contextually grounded responses. Kivimäki (2025) confirmed that frameworks such as Ollama have substantially reduced the technical complexity of local deployment, while Puhakka (2025) showed that integrating Ollama with OpenWebUI as a management interface creates an increasingly mature and accessible local deployment ecosystem. Mienye *et al*. (2025) added that in-context learning and RAG techniques allow smaller local models to approach the performance of larger models in specific domains — further strengthening the case for hybrid deployment as the practical default for most commercial desktop LLM applications.

4.2.5 Theoretical and Practical Implications

Theoretically, this study contributes to the understanding of factors that determine the success of LLM integration in desktop applications, addressing the gap identified by Hou *et al*. (2024) in their systematic review of LLMs in software engineering — a review that did not specifically address the desktop context. The findings show that desktop LLM integration has distinct characteristics that differentiate it from LLM integration in web or mobile applications, particularly regarding deployment considerations, computational resource management, and user expectations around privacy and control. Wang *et al*. (2025) and Han *et al*. (2024) provide the theoretical foundation for understanding the ongoing LLM evolution, while this study adds an applied dimension focused on the desktop as a specific deployment environment with its own constraints and affordances. On the practical side, four actionable guidelines emerge directly from the findings. First, model selection should be based on a comprehensive needs analysis that simultaneously considers required output quality, target hardware constraints, privacy requirements, and operational budget — treating these as interdependent variables rather than independent checkboxes. Second, RAG as the primary integration technique has been consistently shown to improve LLM output accuracy and relevance in domain-specific contexts without the cost of full fine-tuning. Third, UX design investment that accounts for the unique characteristics of LLM output — including mechanisms for building user trust and managing expectations around output uncertainty — is a critical factor that implementations focused solely on technical performance tend to overlook. Fourth, a hybrid deployment strategy combining local models for routine tasks with cloud API access for complex reasoning tasks offers the most balanced trade-off between cost, privacy, and quality for most commercial desktop application scenarios.

4.2.6 Research Limitations

Several limitations must be acknowledged transparently to provide appropriate context for interpreting these findings. First, although searches were conducted across seven databases, the possibility of relevant literature not indexed in those sources cannot be fully ruled out. Second, the search period ending in December 2024 means that developments in the LLM field occurring after that date are not captured in the synthesis — a significant limitation given the pace of change in this area. Third, the methodological heterogeneity of included studies, ranging from controlled experiments to descriptive case studies, limits the feasibility of more rigorous quantitative meta-analysis. Fourth, the majority of included studies originate from institutions in the United States, China, and Europe, which means findings may not fully represent the desktop application development context in other regions, including Southeast Asia. Scherbakov *et al*. (2025) and Susnjak (2023) both pointed to LLM-assisted systematic review processes as a partial solution to some of these limitations in future work — particularly the capacity to process larger literature corpora more efficiently and to support living reviews that update incrementally as new evidence accumulates.

5 | CONCLUSIONS

This study conducted a Systematic Literature Review of 42 articles selected from an initial pool of 1,200 using PRISMA 2020 methodology, to answer four research questions on the integration of Large Language Models in desktop applications. Four principal conclusions emerge from the synthesis, each reinforcing the others. GPT-4 and the LLaMA family are the most dominant LLM architectures integrated in desktop applications, representing the two primary paradigms of proprietary API-based models and open-source locally deployed models respectively. Architectural choice is consistently determined by model parameter size: models exceeding 70 billion parameters tend to be accessed via cloud APIs, while models below 13 billion parameters are more frequently deployed locally using tools such as Ollama and LM Studio.

Computational resource constraints (81.0%), response latency (69.0%), and data privacy and security (57.1%) stand as the three most serious technical challenges in desktop LLM integration. Model quantization techniques, RAG implementation, and streaming output architecture have proven the most workable solutions for addressing these barriers — none of which requires abandoning the desktop deployment model entirely.

LLM integration broadly produces positive effects on User Experience, with 73.8% of studies reporting measurable improvements, particularly in task efficiency and user satisfaction. Output predictability and user trust, however, remain the most challenging UX dimensions and require more deliberate design attention — especially in desktop applications used for critical decision-making where the cost of unreliable output is high.

No deployment strategy is universally superior. The hybrid approach — combining lightweight local models for routine tasks with cloud API access for tasks requiring complex reasoning — has emerged as the most common production practice, offering the most balanced trade-off between output quality, data privacy, operational cost, and offline capability. These findings confirm that LLM integration in desktop applications has moved beyond the experimental phase and entered a stage of mature production adoption. For future research, two directions are particularly warranted: more rigorously controlled empirical studies measuring the long-term effects of LLM integration on user productivity across diverse desktop application domains, and the development of a standardized evaluation approach specifically designed to assess LLM integration quality in the desktop setting — one that accounts for the unique deployment, performance, and UX characteristics that distinguish desktop environments from web and mobile platforms.

REFERENCES

Abdulla, H., AlJazeeri, F., Hewahi, N., & El-Medany, W. (2025, November). *Developing a tailored framework for systematic literature reviews in machine learning (ML-SLR): Addressing challenges and advancing research*. In 2025 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) (pp. 1–6). IEEE. https://doi.org/10.1109/3ICT68299.2025.11442228

Agrawal, S., Oza, P., Kakkar, R., Tanwar, S., Jetani, V., Undhad, J., & Singh, A. (2024). Analysis and recommendation system-based on PRISMA checklist to write systematic review. *Assessing Writing*, *61*, Article 100866. https://doi.org/10.1016/j.asw.2024.100866

Buckley, T. A., Crowe, B., Abdulnour, R. E. E., Rodman, A., & Manrai, A. K. (2025, March). Comparison of frontier open-source and proprietary large language models for complex diagnoses. *JAMA Health Forum*, *6*(3), Article e250040. https://doi.org/10.1001/jamahealthforum.2025.0040

Carrera-Rivera, A., Ochoa, W., Larrinaga, F., & Lasa, G. (2022). How to conduct a systematic literature review: A quick guide for computer science research. *MethodsX*, *9*, Article 101895. https://doi.org/10.1016/j.mex.2022.101895

Castillo, S., & Grbovic, P. (2022). The APISSER methodology for systematic literature reviews in engineering. *IEEE Access*, *10*, 23700–23707. https://doi.org/10.1109/ACCESS.2022.3148206

Chamarthi, B., Polu, O., Anumula, S., et al. (2025, April 22). Natural language processing (NLP)- and machine learning (ML)-enabled operating room optimization: A Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) systematic review anchored in project planning theory. *Cureus*, *17*(4), Article e82796. https://doi.org/10.7759/cureus.82796

Chkirbene, Z., Hamila, R., Gouissem, A., & Devrim, U. (2024, December). Large language models (LLM) in industry: A survey of applications, challenges, and trends. In *2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life Using AI, Robotics and IoT (HONET)* (pp. 229–234). IEEE. https://doi.org/10.1109/HONET63146.2024.10822885

Choi, W. C., & Chang, C. I. (2025). *Advantages and limitations of open-source versus commercial large language models (LLMs): A comparative study of DeepSeek and OpenAI's ChatGPT*. Preprints.org. https://doi.org/10.20944/preprints202503.1081.v1

Han, S., Wang, M., Zhang, J., Li, D., & Duan, J. (2024). A review of large language models: Fundamental architectures, key technological evolutions, interdisciplinary technologies integration, optimization and compression techniques, applications, and challenges. *Electronics*, *13*(24), Article 5040. https://doi.org/10.3390/electronics13245040

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., … Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, *33*(8), 1–79.

Kitchenham, B., Madeyski, L., & Budgen, D. (2022). SEGRESS: Software engineering guidelines for reporting secondary studies. *IEEE Transactions on Software Engineering*, *49*(3), 1273–1298. https://doi.org/10.1109/TSE.2022.3174092

Kivimäki, T. (2025). *Usability evaluation of the local large language models* [Master's thesis, University of Turku].

Kukreja, S., Kumar, T., Purohit, A., Dasgupta, A., & Guha, D. (2024, January). A literature survey on open source large language models. In *Proceedings of the 2024 7th International Conference on Computers in Management and Business* (pp. 133–143). https://doi.org/10.1145/3647782.3647803

Liu, J. (2025, October). AI-powered automated and remote UX evaluation methods: A systematic literature review. In *Proceedings of the 43rd ACM International Conference on Design of Communication* (pp. 10–16). https://doi.org/10.1145/3711670.3764614

Long, H. A., French, D. P., & Brooks, J. M. (2020). Optimising the value of the Critical Appraisal Skills Programme (CASP) tool for quality appraisal in qualitative evidence synthesis. *Research Methods in Medicine & Health Sciences*, *1*(1), 31–42. https://doi.org/10.1177/2632084320947559

Mienye, I. D., Jere, N., Obaido, G., Ogunruku, O. O., Esenogho, E., & Modisane, C. (2025). Large language models: An overview of foundational architectures, recent trends, and a new taxonomy. *Discover Applied Sciences*, *7*(9), Article 1027. https://doi.org/10.1007/s42452-025-07668-w

Mohammad, A., & Chirchir, B. (2024). Challenges of integrating artificial intelligence in software project planning: A systematic literature review. *Digital*, *4*(3), 555–571. https://doi.org/10.3390/digital4030028

Patel, S., & Dholakiya, R. A. (2025, August). Large language models: Evolution, architecture, applications, and future horizons. In *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)* (pp. 1922–1929). IEEE. https://doi.org/10.1109/ICSCSA66339.2025.11170884

Phillips, M., Reed, J. B., Zwicky, D., & Van Epps, A. S. (2024a). A scoping review of engineering education systematic reviews. *Journal of Engineering Education*, *113*(4), 818–837. https://doi.org/10.1002/jee.20549

Phillips, M., Reed, J. B., Zwicky, D., Van Epps, A. S., Buhler, A. G., Rowley, E. M., … Zakharov, W. (2024b). Systematic reviews in the engineering literature: A scoping review. *IEEE Access*, *12*, 62648–62663. https://doi.org/10.1109/ACCESS.2024.3394755

Prabhune, S., & Berndt, D. J. (2024). *Deploying large language models with retrieval augmented generation*. arXiv. https://doi.org/10.48550/arXiv.2411.11895

Puhakka, O. (2025). *Usability of local large language models and retrieval augmented generation in health care* [Bachelor's thesis, University of Oulu]. https://urn.fi/URN:NBN:fi:oulu-202505083171

Scherbakov, D., Hubig, N., Jansari, V., Bakumenko, A., & Lenert, L. A. (2025). The emergence of large language models as tools in literature reviews: A large language model-assisted systematic review. *Journal of the American Medical Informatics Association*, *32*(6), 1071–1086. https://doi.org/10.1093/jamia/ocaf063

Schillaci, Z. (2024). LLM adoption trends and associated risks. In *Large language models in cybersecurity: Threats, exposure and mitigation* (pp. 121–128). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-54827-7_13

Susnjak, T. (2023). *PRISMA-DFLLM: An extension of PRISMA for systematic literature reviews using domain-specific finetuned large language models*. arXiv. https://doi.org/10.48550/arXiv.2306.14905

Wang, S., Liu, W., Chen, J., Zhou, Y., Gan, W., Zeng, X., … Hao, J. (2024). *GUI agents with foundation models: A comprehensive survey*. arXiv. https://doi.org/10.48550/arXiv.2411.04890

Wang, Z., Chu, Z., Doan, T. V., Ni, S., Yang, M., & Zhang, W. (2025). History, development, and principles of large language models: An introductory survey. *AI and Ethics*, *5*(3), 1955–1971. https://doi.org/10.1007/s43681-024-00583-7

Zhang, C., He, S., Qian, J., Li, B., Li, L., Qin, S., … Zhang, Q. (2024). *Large language model-brained GUI agents: A survey*. arXiv. https://doi.org/10.48550/arXiv.2411.18279

Zheng, Y., Chen, Y., Qian, B., Shi, X., Shu, Y., & Chen, J. (2025). A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, *57*(8), 1–35. https://doi.org/10.1145/3719664.

How to cite this article: Mokoginta, D. (2026). Large Language Model Integration in Desktop Applications: A Systematic Literature Review. *Journal Dekstop Application (JDA)*, *5*(1), 1-16. https://doi.org/10.59431/jda.v5i1.810.