

Why Industry Migrates and When It Trusts LLMs: A Practitioner Survey on LLM-Assisted Software Migration

Sivajeet Chand ✉️🏠 

Technical University of Munich, Munich, Germany

Gedeon Lenz ✉️ 

CHECK24, Munich, Germany

Patrick Funke ✉️ 

CHECK24, Munich, Germany

Derui Zhu ✉️🏠 

Technical University of Munich, Munich, Germany

Alexander Pretschner ✉️🏠 

Technical University of Munich, Munich, Germany

Abstract

Software migrations are frequent and high-stakes industrial activities. Although large language models (LLMs) are increasingly used in software development, their suitability for migration remains uncertain because migration requires semantic preservation, project-specific context, security, and production accountability. We investigate why practitioners undertake software migrations, what motivates and constrains the use of LLMs for migration, where LLM assistance is considered acceptable, and which factors determine trust in LLM-migrated code. We conducted a practitioner survey with 61 retained industry respondents, combining an internal industrial sample with an open professional survey. We analyzed closed and Likert-style items quantitatively and interpreted open-ended responses thematically. Migration was primarily framed as a bundled quality-risk activity: maintainability, scalability, and security were the dominant motives, while cost reduction rarely appeared as a standalone driver. Respondents valued LLMs mainly as accelerators for exploration, prototyping, and routine transformation, but not as autonomous migration agents. Acceptability was strongly bound by scope and control: 62.3% considered LLMs acceptable most of the time or almost always for single-file migration, compared with only 4.9% for entire-project migration. Trust depended chiefly on semantic correctness, project-context awareness, security, verification evidence, and provenance disclosure. Practitioners do not reject LLM-assisted migration outright; rather, they adopt a conditional trust model. LLMs are viewed as useful migration accelerators, but acceptable production use depends on bounded scope, human review, assurance evidence, traceability, and clear accountability.

2012 ACM Subject Classification Software and its engineering → Software maintenance tools; Software and its engineering → Maintaining software; Software and its engineering → Software evolution; Software and its engineering → Software organization and properties

Keywords and phrases LLMs, Code Migration, Industrial Insights, LLM-Assisted, Code Translation

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.85

Category Software Engineering in Practice Track Paper

Supplementary Material *Dataset*: https://figshare.com/articles/dataset/Why_Industry_Migrates_and_When_It_Trusts_LLMs_Data/32414763?file=64973589



© Sivajeet Chand, Gedeon Lenz, Patrick Funke, Derui Zhu, and Alexander Pretschner; licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 85; pp. 85:1–85:21



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Software migration is a central activity in software maintenance and evolution, enabling systems to remain maintainable, scalable, and secure amid changing requirements and ecosystems. Industrial migrations, such as language transitions, framework upgrades, or architectural shifts, are often large-scale, costly, and error-prone, as they require extensive code transformation while preserving system behavior and operational constraints [14, 12, 23].

In parallel, large language models (LLMs) have rapidly emerged as a powerful paradigm for software engineering tasks. Recent work has demonstrated their effectiveness in code generation, refactoring, bug fixing, and documentation support [5, 24, 6]. These capabilities have led to widespread adoption of LLM-based tools in industrial development workflows, where they are primarily used to accelerate coding tasks and reduce repetitive effort [6, 40, 28].

More recently, both research and practice have begun exploring the use of LLMs for code transformation and migration tasks, such as translating code across programming languages, adapting APIs, or restructuring legacy code [24, 5, 10]. While these approaches show promising results in controlled settings, they typically focus on functional correctness at the snippet or file level, often evaluated through benchmarks or small-scale experiments [24, 26, 33, 36]. Existing repository-level translation studies show that performance on fine-grained translation tasks does not directly scale to full repositories. While prior approaches improve scalability by decomposing programs into smaller translation units and using dependencies primarily to determine translation order, they often translate interdependent components in isolation. This can lead to integration inconsistencies, as strong fragment-level translation accuracy does not guarantee correctness at the repository level, where inter-file dependencies, build constraints, and system-level validation are essential [35, 16, 31, 38].

Industrial software migration introduces requirements that go beyond these settings. Migration tasks must preserve semantic behavior across complex systems, maintain compatibility with existing architectures, adhere to organizational coding standards, and satisfy strict quality and security constraints. In this context, even small deviations, such as mishandled edge cases or incorrect assumptions about system context, can lead to significant failures in production. As a result, the applicability of LLMs to real-world migration remains uncertain. This creates a fundamental tension: LLMs promise productivity gains and reduced effort, yet industrial migration demands high trust, reliability, and verifiability. Despite growing interest in LLM-assisted development, there is limited empirical evidence on how practitioners perceive and adopt LLMs for software migration in real-world settings. Existing research has largely focused on model capabilities, while questions of trust, adoption boundaries, acceptable usage scope, and perceived risks remain underexplored, particularly in industrial contexts. Understanding these factors is essential, as they ultimately determine whether and how LLM-based approaches can be integrated into production migration workflows.

This study was motivated by industrial migration experience at CHECK24, where practitioners had performed large-scale software migration in production systems and encountered the practical question of whether, where, and under what conditions LLMs could support such work. These experiences highlighted a broader industrial problem: LLMs may accelerate migration tasks, but migration work also requires semantic preservation, project-specific context, security, reviewability, and accountability. To investigate this problem systematically, we conduct a practitioner survey on LLM-assisted software migration, combining responses from CHECK24 practitioners with an open survey of developers across organizations. This design allows us to capture both context-rich insights from a concrete industrial migration setting and broader practitioner perspectives on the motivations, risks, and trust conditions surrounding LLM-assisted migration. We structure our investigation around the following research questions:

RQ1 (Migration intentions). What are the intentions (business and engineering) driving software migrations in industry, and how do these vary by role, production exposure, and migration type?

This RQ characterizes why organizations initiate migrations (e.g., maintainability, scalability, security, cost, ecosystem constraints). It also examines whether prioritization of these intentions differs across practitioner roles, exposure to production migrations, and the categories of migrations performed.

RQ2 (Motivations for using LLMs). What motivates practitioners to use LLMs for migration work, and what conditions (risk perceptions, required assurance, review effort) gate their willingness to use LLM-generated migration code in production?

This RQ captures the perceived value proposition of LLMs for migration tasks (e.g., productivity, reduced barrier of entry, faster exploration) and the factors that constrain adoption. It focuses on production readiness by eliciting the assurances, verification evidence, and review processes required before accepting LLM outputs.

RQ3 (Acceptability scope and conditions). Where (and under what conditions) are LLMs acceptable for migration tasks?

This RQ operationalizes acceptable usage boundaries by studying which migration granularities practitioners tolerate (e.g., small parts, single files, entire projects) and how this changes between non-production and production contexts. It also covers attitudes toward partial versus full automation and the expected role of human oversight.

RQ4 (Trust determinants). What risks, failure modes, and quality expectations determine trust in LLM-migrated code?

This RQ identifies the principal sources of distrust (e.g., semantic drift, mishandled edge cases, security concerns, increased review burden) and the quality attributes practitioners consider non-negotiable for migrated code. It relates these expectations to perceived impacts on key outcomes such as code quality, maintainability, security, and review effort.

Our results reveal a consistent pattern across both the company-internal and open-survey samples. Practitioners recognize LLMs as useful tools for supporting localized migration tasks, such as transforming small components or accelerating exploratory work. However, they express strong reservations regarding fully automated migration, particularly in production environments. Instead, developers favor partial automation combined with human review, reflecting a cautious, control-oriented adoption model. Trust in LLM-generated migration code is primarily shaped by concerns about semantic correctness, handling of edge cases, integration with system-specific context, and security implications. Moreover, respondents tend to treat LLM-generated migration code as draft material requiring substantial review rather than as production-ready output.

In summary, this paper makes the following contributions:

- Empirical evidence on how industrial practitioners perceive LLM-assisted code migration.
- A characterization of acceptable usage boundaries, highlighting the role of granularity and human oversight.
- An analysis of trust determinants, including risks, failure modes, and quality expectations.
- Actionable insights into adoption barriers and enabling conditions for integrating LLMs into migration workflows.

By grounding LLM-assisted migration in practitioner perspectives, this work contributes to a more realistic understanding of how such techniques can be applied in industrial software migration.

2 Related Work

Software migration and code translation have long been studied in software maintenance and evolution, including concrete settings such as Java-to-Kotlin migration and C-to-Rust translation, where migration must preserve behaviour, integrate with the target ecosystem, and satisfy quality and safety constraints beyond syntactic rewriting [23, 14, 12]. Neural and LLM-based translation work has shown both the feasibility and limitations of automation: early approaches demonstrated unsupervised program translation [26], while recent benchmarks and empirical studies report semantic mistakes, edge-case failures, and translation bugs in multilingual translation [36, 24, 33, 37, 7]. Related work on API and library migration shows that LLMs can support more specific migration tasks, but correctness and validation remain central challenges [39, 2].

Recent work has moved from isolated translation tasks to repository-, system-, and industrial-scale migration, showing that local translation accuracy does not directly scale to whole systems where inter-file dependencies, build constraints, and cross-component consistency become central [16, 35, 38]. To improve validation, other approaches combine LLMs with tests, static or dynamic analysis, call graphs, or human feedback [27, 31, 21, 10]. Industrial experience reports, including large-scale LLM-assisted migration at Google, demonstrate practical potential while still relying on review, validation, and developer oversight [40].

Recent practitioner studies have examined how developers use AI and GenAI tools for code-related tasks. Wang et al. show that practitioners expect code completion to be correct, context-aware, controllable, and usable [34], while Liang et al. find that AI programming assistants are valued for reducing effort and accelerating implementation, but remain difficult to steer when suggestions violate functional or non-functional expectations [19]. Li et al. further show that adoption is shaped not only by individual productivity motives, but also by organizational factors such as policy, privacy, training, and knowledge sharing [18]. Recent studies also identify risks around GenAI-generated code, including licensing, provenance, ownership, and long-term effects on expertise and accountability [32, 8]. Overall, this work shows that practitioners view AI coding assistants as useful but conditional support tools; however, it mainly studies general coding, code completion, productivity, adoption, or legal concerns. To the best of our knowledge, prior survey work has not examined LLM-assisted software migration as an industrial trust and acceptability problem. We address this gap by combining internal and broader practitioner responses to characterize the conditions under which LLMs are viewed as useful, acceptable, and trustworthy for production migration workflows, rather than evaluating a specific migration technique.

3 Methodology

We designed a survey to address the research questions introduced in Section 1, following established guidelines for survey research in software engineering [25, 20, 29, 30]. The survey aimed to examine how software practitioners with varying levels of experience perceived, used, and trusted LLMs in software migration tasks. We adopted a mixed-methods design: closed and Likert-style questions collected structured evidence on migration experience, LLM usage, perceived usefulness, acceptability boundaries, quality expectations, and trust, while open-ended questions captured explanations, contextual constraints, and concerns not covered by predefined response options.

3.1 Survey Instrument

We iteratively developed the survey instrument (Table 1) from literature-derived constructs aligned with our research questions. The survey included closed, multi-select, Likert-style, top- k selection, and open-ended questions. Likert-style items measured degree-based constructs, including self-rated experience, familiarity with LLMs, importance of migration drivers and quality expectations, and perceived LLM impact on software-engineering outcomes. Open-ended questions followed selected structured items to elicit explanations and additional themes. To ensure practical relevance, we conducted formative scoping interviews with six practitioners from CHECK24, one of the largest comparison-platform companies in Germany. Their feedback on migration challenges and LLM-use concerns informed the questionnaire’s structure, wording, and response options. The interviews were used only to improve construct validity and real-world applicability and were not analyzed or reported as an independent empirical dataset.

■ **Table 1** Survey questions mapped to research questions with measured constructs. Legend: C = Closed, L = Likert, O = Open-ended, B = Both.

RQ	Question	Measured Construct	Type
RQ0	Q1: Primary role (Developer, Manager)	Practitioner role	C
	Q2: Role experience (Developer/Manager experience)	Seniority	C
	Q3: Have you migrated a project/system?	Migration exposure	C
	Q4: Type(s) of migration performed	Migration expertise	C
	Q5: Self-rated migration experience	Migration expertise	L
	Q6: Familiarity with LLMs	LLM awareness	L
	Q7: Used LLMs for work before	LLM exposure (work)	C
	Q8: Experience using LLMs for coding	LLM coding usage	C
	Q9: Experience using LLMs for migrations	LLM migration usage	C
	Q12: LLMs used for work	LLM usage (work)	C
	Q13: LLMs used in general	LLM usage (general)	C
	Q14: Preferred LLM for coding	LLM preference	C
	Q15: Coding assistants used for work	Assistant usage (work)	C
	Q16: Coding assistants used in general	Assistant usage (general)	C
	Q17: Most frequently used coding assistant	Assistant preference	C
	Programming language background	Technical ecosystem	C
RQ1	Q10: Migration intentions (e.g., scalability, maintainability, cost)	Migration drivers	L
	Q11: Other migration intentions (free-text)	Additional drivers	O
RQ2	Q18: Contexts of LLM usage (general, coding, migration)	Usage contexts	C
	Q19: Drivers of LLM adoption in workplace	Adoption motivation	C
	Q28: Workflow advantages using LLMs	Workflow benefits	B
	Q29: Advantages in LLM-generated migration code	Migration-specific benefits	B
	Q37: Adoption hurdles of LLMs	Barriers and constraints	C
RQ3	Q20: Acceptability by migration granularity	Scope acceptance	C
	Q21: Reasoning for acceptability (free-text)	Justification	O
	Q22: More critical in production vs non-production	Production sensitivity	C
	Q23: Comfort reviewing full-project LLM migrations	Review confidence	C
	Q24: Willingness to improve LLM migration systems	Engagement readiness	C
	Q25: Stance on automation (full vs partial)	Automation preference	C
	Q38: Willingness to actively support LLM migration adoption	Adoption participation	C
	Q39: Confidence reviewing full-project migration before deployment	Deployment readiness	C
	Q40: Willingness to deploy automated migrations in production	Production deployment stance	C
	Q41: Acceptance of full automation	Automation stance	C
	Q42: Acceptance of partial automation	Automation stance	C
RQ4	Q26: Workflow issues when using LLMs	Workflow failure modes	B
	Q27: Issues in LLM-generated migration code	Migration failure modes	B
	Q30: Importance of quality expectations	Required quality attributes	L
	Q31: Additional expectations (free-text)	Extended quality attributes	O
	Q32: Deviating standards for LLM vs manual migration	Standard differences	O
	Q33: Whether LLMs or humans are held to higher standards	Trust baseline	C
	Q34: Trust depending on code source and disclosure	Source-based trust	C
	Q35: Importance of outcome dimensions	Outcome importance	L
	Q36: Perceived impact of LLMs on outcomes	Outcome impact	L
	Q37: Adoption hurdles affecting trust	Trust barriers	C

The questionnaire consisted of five parts. First, we collected participant context, including role, professional experience, migration experience, familiarity with LLMs, prior use of LLMs and coding assistants, and programming-language background, to characterize the sample and contextualize later responses. Second, we examined software migration experience and intentions. Respondents rated the relevance of common business and engineering drivers, including maintainability, scalability, cost reduction, security, and ecosystem constraints, and provided open-ended responses for additional intentions not covered by predefined options. Third, we assessed LLM usage and adoption, including general and workplace use of LLMs and coding assistants, use for coding and migration, perceived adoption drivers, workflow advantages, and migration-specific advantages of LLM-generated code. Fourth, we evaluated the acceptability of LLM assistance for migration tasks across granularities, including single files, small system components, and entire projects. We also examined production sensitivity, confidence in reviewing LLM-migrated projects, willingness to support LLM-based migration adoption, and attitudes toward partial and full automation. Fifth, we investigated trust, risks, failure modes, and quality expectations, including workflow issues, migration-specific issues in generated code, required quality attributes, differences between LLM-performed and manual migrations, source-based trust, and perceived impacts on productivity, maintainability, security, code quality, and review effort.

3.2 Participants and Sampling

The target population comprised software practitioners with experience in software development, software migration, LLM-assisted development, or combinations thereof. We collected responses from two complementary non-probabilistic samples. The first was an internal industrial sample from CHECK24, selected through purposive sampling: practitioners were invited because they had participated in industrial software migration work, including language migration in production systems. This provided first-hand migration experience from an organizational setting. The second was an open survey of professional developers outside the internal organization, recruited through convenience sampling via the authors' professional networks, primarily LinkedIn and related contacts, to broaden the study beyond one organization and capture other industrial perspectives. Accordingly, we do not claim statistical representativeness; rather, the samples provide empirically grounded evidence on practitioner perceptions, adoption conditions, and trust boundaries for LLM-assisted software migration.

3.3 Data Collection

Data were collected between January 2026 and April 2026. Accordingly, the findings represent practitioner perceptions during this period and should be interpreted in the context of the LLM capabilities and tooling available at that time.

We invited 73 practitioners from the internal organization who have migration experience. The internal practitioners survey yielded $N = 25$ responses. The open survey yielded $N = 42$ raw responses. The internal sample was analyzed as an industrial migration-practitioner sample. The open sample was filtered to retain respondents who were employed industry developers and provided usable survey data. Because the open survey was distributed through LinkedIn and professional contacts, the total number of recipients was not known, and a response rate could not be computed. Prior to participation, respondents were informed that the survey was strictly anonymous, required no personally identifiable information, and would take approximately 15 minutes. While the questionnaire was presented in English,



■ **Figure 1** Filtering process for the full survey sample.

free-text responses were accepted in either English or German. Participants were encouraged to answer precisely or provide custom feedback if predefined options were inadequate. Finally, any voluntarily disclosed company affiliations are used solely to demonstrate sample diversity, ensuring no identifiable organizational data is reported.

3.4 Data Filtering and Preparation

We applied a two-step conservative filtering strategy to the full set of collected responses. The first filtering step checked target-population membership. For the internal sample, this criterion was satisfied by design because practitioners were invited based on involvement in industrial software migration. For the open survey, we explicitly applied the same criterion and removed four responses from participants who were not currently employed as industry developers and provided no substantive answers beyond the screener, reducing the open-survey sample from $N = 42$ to $N = 38$. The second step screened response-quality risks using completion time. Among screened responses, the mean completion time was 18.76 minutes, the median was 11.58 minutes, and the 5th percentile was 5.23 minutes. We therefore used completion time below five minutes as a conservative threshold and applied it to both samples. Two open-survey responses fell below this threshold and were excluded; one also showed straight-lining across Likert-style items, with all detected Likert-style answers marked neutral. No internal response fell below the five-minute threshold. After filtering, the final dataset comprised $N = 61$ practitioner responses: $N = 36$ open-survey respondents and $N = 25$ internal respondents. Respondents were not excluded solely for lacking migration experience; instead, migration experience was used as a contextual variable, enabling comparisons between migration-experienced respondents and other industry developers while retaining data on general LLM adoption, perceived trust, and acceptability. Figure 1 summarizes the two-step filtering process for the full survey sample, distinguishing the open-survey and internal samples.

For subgroup analyses, screening questions classified respondents by general, production, or migration experience. Closed, multi-select, and Likert-style items were encoded as categorical, binary, and ordinal variables. Since Likert intervals were treated as unequal, we analyzed them using medians, interquartile ranges, and distributions; top-k questions by frequency; and valid open-ended/custom responses as coding units for thematic analysis.

3.5 Sample Coverage and Scope of Inference

The final dataset comprises $N = 61$ valid practitioner responses, including an open survey sample ($N = 36$) and an internal sample ($N = 25$). It provides empirical coverage of the target phenomenon: 57 respondents had participated in software migrations, 46 with production-environment experience, and 32 had used LLMs for migration tasks. Although non-probabilistic and not statistically representative of the broader population, this complementary sample offers sufficient depth for the exploratory and descriptive aims of this study.

3.6 Quantitative Analysis

We analyzed categorical data using frequencies and percentages, and evaluated ordinal Likert responses through distributions, medians, and interquartile ranges.

For within-respondent comparisons, we employed Friedman tests [11] (effect size: Kendall's τ [17]) followed by Holm-corrected Wilcoxon signed-rank post-hoc tests [13]. We assessed exploratory between-group differences via Mann-Whitney U tests (effect size: rank-biserial correlations) [22, 9], interpreting these as indicative trends rather than population-level effects due to limited sample sizes. Finally, to address RQ2, we fitted an exploratory logistic regression, reporting odds ratios, 95% CIs, and p-values, to examine factors influencing perceived LLM helpfulness without inferring causality [1, 15].

Additionally, we systematically evaluated the research questions from distinct perspectives. RQ1 examined migration intentions across practitioner backgrounds; RQ2 analyzed LLM usage scenarios, adoption drivers, and barriers in migration tasks; RQ3 assessed developer acceptability and review confidence across migration granularities and automation levels; and RQ4 examined failure modes, trust mechanisms, quality expectations, and perceived impacts on code quality, security, and productivity.

3.7 Qualitative Analysis

We analyzed open-ended responses using thematic analysis [3, 4] via a four-step process: data familiarization, open coding, theme extraction (e.g., semantic drift, review burden), and mapping to research questions. Using a hybrid inductive-deductive approach guided by core survey constructs, two authors independently coded an initial subset, resolved disagreements to standardize a codebook, and then analyzed the full sample. The resulting qualitative themes were used to explain and contextualize our quantitative findings.

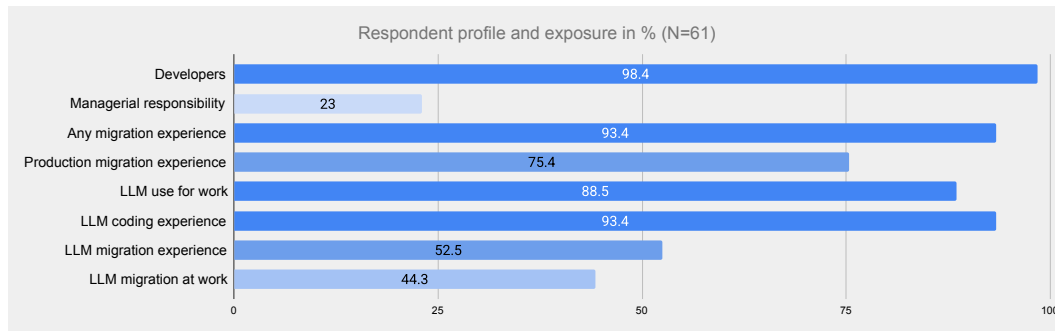
3.8 Integration of Quantitative and Qualitative Results

This study integrates quantitative and qualitative results to provide a comprehensive analysis. Quantitative data identified dominant response patterns, while qualitative feedback explained the underlying reasoning. For instance, the preference for LLM assistance in small migration tasks over full-project automation was driven by concerns regarding semantic preservation, missing context, security, and increased review effort. This mixed-methods approach moves beyond merely reporting frequencies, ultimately revealing the trust boundaries that shape practitioners' acceptance of LLM-assisted migration.

4 Results

We received 25 responses from the internal invitation wave sent to 73 practitioners and 42 responses from the external open survey. After filtering (Figure 1), the final analysis-ready sample comprised 61 practitioners: 25 internal respondents and 36 external respondents. The internal response rate was 34.2%; an external response rate could not be computed because the open survey was distributed publicly rather than through an invite-tracked sampling frame.

As presented in Figure 2 final sample was strongly practitioner-centric. Of the 61 respondents, 60 identified as developers and 14 also reported managerial responsibilities. Migration experience was widespread: 57 respondents reported some migration experience and 46 reported production migration experience. Prior LLM exposure was also high, with 54 respondents reporting LLM use for work, 57 for coding tasks, and 32 specifically for migration-related tasks. These distributions indicate that the sample is suitable for analysing practitioner experience with both software migration and LLM-supported development.



■ **Figure 2** RQ0: Respondent profile and exposure to migration/LLM-assisted development.

This baseline matters for interpreting the remainder of the study. The results should not be read as novice or purely speculative opinions about LLMs. Rather, the dataset predominantly captures views from respondents who have both migration exposure and practical familiarity with LLM-assisted coding, making later differences in acceptance and trust particularly informative.

4.1 RQ1: Migration Intentions

As shown in Figure 3a, migration was most often associated with maintainability, scalability, and security: 70.5%, 65.6%, and 62.3% of respondents rated these drivers as important, respectively. Missing features in the previous ecosystem or language were rated important by 32.8%, and cost reduction by 24.6%. Thus, in this sample, migration was framed primarily as a quality, evolvability, and risk-management activity rather than as cost optimisation.

The item-level responses further suggest that these drivers were bundled rather than isolated. Across the retained sample, 58/61 respondents (95.1%) rated at least one of maintainability, scalability, or security as important; 45/61 (73.8%) rated at least two; 18/61 (29.5%) rated all three; and only 3/61 (4.9%) rated none of them as important. Cost reduction was also non-isolated: although 15/61 respondents (24.6%) rated it as important, all of them also rated at least one of maintainability, scalability, or security as important. Similarly, among the 20 respondents who rated missing features as important, 19 also rated at least one of these three quality- and risk-oriented drivers as important. Cost and capability gaps therefore appeared as part of a broader evolvability and risk profile rather than as standalone migration motives.

The within-respondent ordering of migration drivers differed significantly across items, Friedman $\chi^2 = 45.88$, $p < 0.001$, Kendall's $W = 0.19$, indicating modest agreement in driver rankings. Maintainability, scalability, and security formed the highest-rated group, with no large separation among them, supporting the view that migration motives cluster as jointly important concerns rather than a sharp hierarchy.

Figure 3b summarizes the company-internal practitioners(internal)–open-survey respondents(external) comparison. Internal respondents rated scalability higher than external respondents, mean 3.80 vs. 3.44, Mann–Whitney $p = 0.004$, rank-biserial effect ≈ 0.36 ; external respondents rated missing ecosystem/language features higher, mean 3.33 vs. 2.76, $p = 0.010$, effect ≈ 0.35 in the opposite direction. These results describe a difference between the CHECK24 sample and the external sample. Production experience may contribute to this pattern, but the groups also differ in organizational context, technology stacks, migration types, roles, and recruitment channels. The comparison is therefore associative and does not establish why the groups rated these drivers differently.

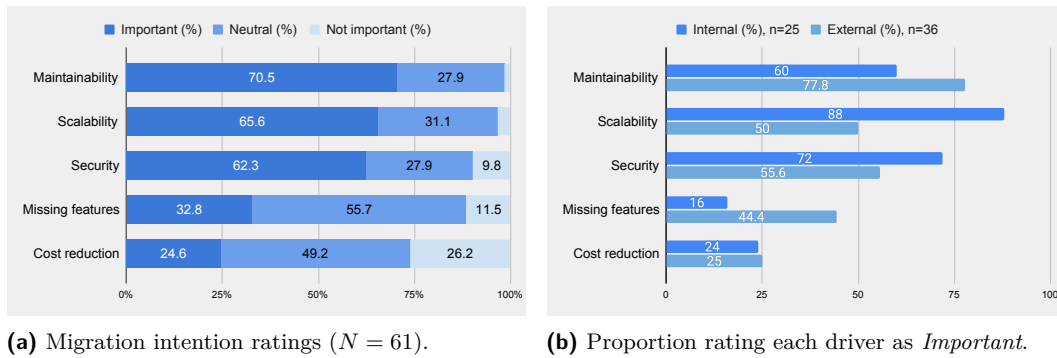


Figure 3 RQ1: Migration intentions.

Migration type was also associated with perceived motives: respondents with infrastructure-migration experience rated scalability higher than those without such experience, mean 3.72 vs. 3.47, $p = 0.043$, effect ≈ 0.25 . This suggests that migration intentions partly depend on the technical locus of change, with infrastructure-oriented migrations particularly associated with scalability concerns.

Qualitative responses added pressures not captured by the Likert items: end-of-life technologies, unsupported frameworks or dependencies, hardware constraints, platform shifts, vendor/provider mandates, and organizational changes in ownership, team responsibility, or governance. Thus, migration intentions extend beyond technical debt or desired quality improvement to ecosystem expiry, external constraints, and organizational restructuring.

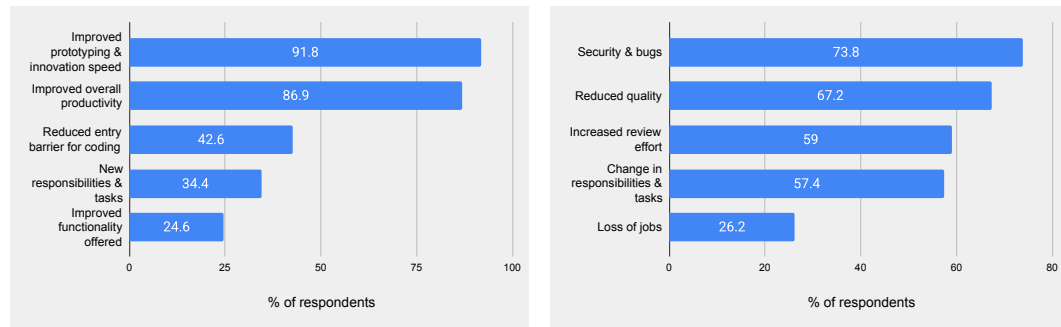
Summary of RQ1. Migration was framed mainly as a bundled quality-risk activity, with maintainability, scalability, and security often co-occurring and cost reduction rarely appearing alone. Motives varied by context: scalability was stronger among internal and infrastructure-migration respondents, while external respondents emphasized ecosystem or language gaps. Qualitative responses added external constraints, ecosystem expiry, and organizational restructuring.

4.2 RQ2: Motivations for using LLMs and gating conditions

Respondents perceived a strong value proposition for LLMs in migration, but mainly as support for acceleration rather than autonomous execution.

As shown in Figure 4a, the leading workplace adoption drivers were improved prototyping and innovation speed, selected by 56/61 respondents (91.8%), improved overall productivity, 53/61 (86.9%), and reduced barriers to unfamiliar coding tasks, 26/61 (42.6%). A separate workflow-advantage item showed the same migration-process pattern: faster migration and prototyping, 48/61 (78.7%); reduced manual effort and repetitive work, 44/61 (72.1%); and lower entry barriers for unfamiliar technologies, 41/61 (67.2%). Together, these results suggest that practitioners value LLMs mainly as accelerators for exploration, prototyping, and first-draft transformation, not as substitutes for human-controlled migration work.

This bounded interpretation is reinforced by context-specific helpfulness ratings. Respondents rated LLMs as more helpful in general and for coding than for migration, Friedman $\chi^2 = 20.18$, $p < 0.001$, Kendall's $W = 0.20$. Post-hoc Wilcoxon tests showed lower helpfulness for migration than for coding, $p = 0.003$, and general use, $p = 0.002$, while general and coding helpfulness did not differ reliably. Thus, respondents did not transfer general confidence in LLM-assisted programming directly to migration, treating it as a more constrained and higher-risk context.



(a) Adoption *drivers*: perceived benefits motivating LLM use in the workplace. (b) Adoption *hurdles*: barriers constraining LLM use.

■ **Figure 4** RQ2: Adoption drivers and hurdles for LLM-assisted migration.

The exploratory logistic regression supports this distinction, but should be interpreted cautiously given the sample size and model specification. Rating LLMs as helpful for migration was strongly associated with having used LLMs for migration at work, $OR = 9.30$, 95% CI [1.99, 43.43], $p = 0.0046$. Managerial responsibility in addition to development was also positively associated with migration helpfulness, $OR = 5.97$, 95% CI [1.19, 30.08], $p = 0.030$, although this result is tentative because the confidence interval is wide and role may be confounded with seniority, decision responsibility, or migration-planning exposure. Generic LLM familiarity was positive but not statistically reliable. Thus, task-specific exposure to LLM-based migration appears more informative for migration-specific helpfulness than general LLM enthusiasm.

However, perceived usefulness did not imply unconditional production acceptance. Respondents imposed strong assurance gates: 46/61 (75.4%) were more critical of LLM-migrated code in production than in non-production settings. For automated LLM migration deployment in production, only 2/61 (3.3%) gave an unconditional “yes”, while 40/61 (65.6%) supported deployment only with sufficient manual review and intervention, and 19/61 (31.1%) rejected it. For reviewing full-project LLM migrations before deployment, only 15/61 (24.6%) felt comfortable, 30/61 (49.2%) did so only under experienced supervision, and 16/61 (26.2%) did not. Adoption is therefore constrained not by categorical rejection of LLM assistance, but by the conditions under which LLM outputs become production-relevant.

Figure 4b summarizes the adoption hurdles behind these gates. The leading barriers were security and bugs, 45/61 (73.8%); reduced quality, 41/61 (67.2%); increased review effort, 36/61 (59.0%); and changes in responsibilities or tasks, 35/61 (57.4%). This reveals a central tension: LLMs may reduce implementation effort while increasing validation, review, and accountability costs. The tension was systematic: all 61 respondents selected at least one positive adoption driver, but 59/61 also selected at least one hurdle. Thus, the key adoption question is not whether LLMs are useful, but under which assurance conditions their outputs can be accepted.

The perceived advantages of LLM-generated migration code also indicate a bounded value proposition. Respondents most often selected code-level benefits related to target-technology conventions and routine migration hygiene: encoding common best practices, 35/61 (57.4%); avoiding common routine-migration mistakes, 30/61 (49.2%); applying standard architectural patterns, 28/61 (45.9%); and producing clear structure, separation of parts, and naming as prompted, 26/61 (42.6%). In contrast, only 8/61 (13.1%) selected less hidden and more explicit behaviour in code. Respondents therefore valued LLM-generated migration code mainly as structured first-draft scaffolding around known conventions, not as evidence of deep semantic assurance.

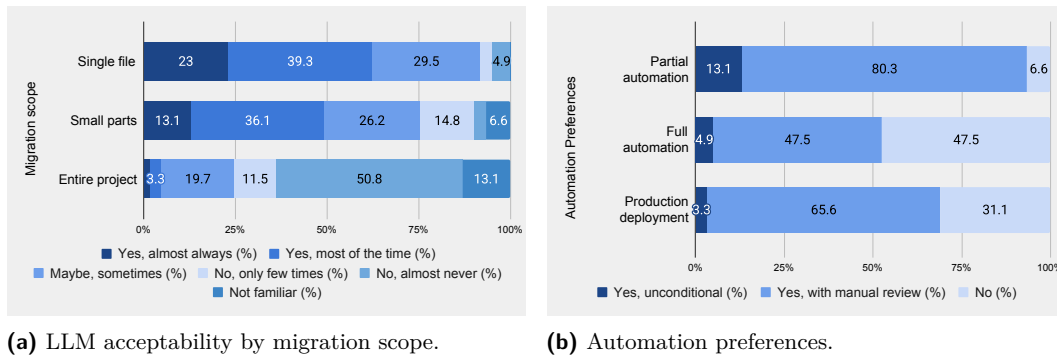


Figure 5 RQ3: Acceptability scope and automation preferences.

Qualitative responses further explain this boundary. Respondents justified caution through large-context confusion, opaque business logic, incomplete knowledge of undocumented systems, and the need for humans to understand and own high-stakes production changes. The main adoption gate is therefore not only whether an LLM can generate plausible migrated code, but whether the resulting changes can be reviewed, understood, secured, and accepted as organisationally accountable.

Summary of RQ2. *Practitioners valued LLMs mainly as migration accelerators: tools for exploration, prototyping, routine transformation, and reducing manual effort. However, perceived usefulness did not imply unconditional production acceptance; adoption remained gated by reviewability, assurance evidence, security, quality, and human accountability.*

4.3 RQ3: Acceptability scope and conditions

As shown in Figure 5a, acceptability declined sharply with migration scope. LLMs were considered acceptable most of the time or almost always by 38/61 respondents (62.3%) for single-file migration and 30/61 (49.2%) for small parts of a system, but only by 3/61 (4.9%) for entire-project migration. For entire-project migration, 38/61 respondents (62.3%) answered either “no, only very few times” or “no, almost never”, 12/61 (19.7%) selected “maybe, sometimes”, and 8/61 (13.1%) selected “not familiar”. Thus, respondents did not reject LLM-assisted migration categorically; rather, acceptability collapsed when the unit shifted from localized code to whole-project transformation.

The scope effect was statistically and practically strong: within-respondent acceptability differed across migration scopes, Friedman $\chi^2 = 74.96$, $p < 0.001$, Kendall’s $W = 0.72$. Post-hoc Wilcoxon tests showed a significant drop from single-file to small-part migration, $p = 0.023$, with a small effect, and much larger drops from small-part to entire-project and from single-file to entire-project migration, both Holm-adjusted $p \approx 4.0 \times 10^{-9}$. Scope therefore emerged as a central boundary condition: practitioners were comparatively open to bounded local assistance, but not to system-wide migration autonomy.

A second boundary concerns automation mode, summarized in Figure 5b. Support was strongest for partial automation: 57/61 respondents (93.4%) endorsed it conditionally or unconditionally, and only 4/61 (6.6%) rejected it. However, support was mainly review-preserving: 49/61 (80.3%) supported partial automation only with sufficient manual review and intervention, while 8/61 (13.1%) supported it unconditionally. Full automation was

weaker: 3/61 (4.9%) supported it unconditionally, 29/61 (47.5%) only with sufficient manual review and intervention, and 29/61 (47.5%) rejected it. Production deployment followed the same pattern: 2/61 (3.3%) supported automated LLM migration deployment unconditionally, 40/61 (65.6%) only with sufficient manual review and intervention, and 19/61 (31.1%) rejected it. Automation preferences differed significantly within respondents, Friedman $\chi^2 = 32.43$, $p < 0.001$, Kendall's $W = 0.27$. Pairwise tests showed stronger support for partial automation than for full automation, $p = 1.17 \times 10^{-6}$, and production deployment, $p = 2.67 \times 10^{-5}$, both with large effects. Full automation and production deployment did not differ reliably, $p = 0.072$, suggesting that respondents treated both as high-stakes escalation points relative to partial, human-controlled automation.

Review capability formed another practical boundary: only 15/61 respondents (24.6%) felt capable of performing the final review of full-project LLM-migrated code before production deployment; 30/61 (49.2%) would do so only under experienced supervision, and 16/61 (26.2%) would not. Thus, full-project migration was rejected not only because generation is difficult, but also because review itself is perceived as difficult, high-risk, and experience-dependent. Internal and external respondents showed the same overall acceptability boundary. External respondents were somewhat more accepting of small-part migration, Mann–Whitney $p = 0.047$, effect ≈ 0.30 , but there was no meaningful internal–external difference for single-file or full-project migration. Given the exploratory subgroup analysis, this difference should be interpreted cautiously. Overall, disagreement concerned where the safe bounded zone ends, not whether whole-project autonomy is broadly acceptable.

Prior LLM migration use did not remove this boundary. Among respondents who had used LLMs for migration at work, 14/27 (51.9%) supported full automation conditionally or unconditionally, compared with 18/34 (52.9%) among others. For production deployment, 19/27 respondents with LLM migration work experience (70.4%) supported deployment conditionally or unconditionally, compared with 23/34 (67.6%) among others. However, none of the experienced respondents supported production deployment unconditionally, and only 1/27 (3.7%) considered entire-project migration acceptable most of the time or almost always. Thus, caution about large-scope autonomy persisted even among respondents with task-specific LLM migration experience. Caution did not imply disengagement. Asked whether they would help drive and optimize automated LLM-based code migration at work, 31/61 respondents (50.8%) would take initiative, 11/61 (18.0%) would participate given an existing initiative, and 15/61 (24.6%) answered maybe if an opportunity arose. Only 2/61 (3.3%) would not participate, and 2/61 (3.3%) were unsure. Respondents were therefore willing to engage with LLM-assisted migration under bounded, reviewable, human-controlled conditions.

Qualitative answers explain this boundary consistently. Respondents associated larger migration scopes with context-window limitations, difficulty recovering implicit business logic from legacy systems, non-standard implementation patterns, weak project-specific knowledge, and higher costs of security or correctness failures. The acceptability boundary is therefore principally a context-and-control boundary rather than a simple capability boundary: practitioners are willing to use LLMs where the task remains local, reviewable, and human-owned, but not where migration becomes system-wide, autonomous, or production-critical.

Summary of RQ3. *Acceptability was strongly bounded by migration scope and automation mode. Respondents were comparatively open to LLM assistance for local, reviewable migration tasks, especially partial automation with manual intervention, but acceptance collapsed for entire-project migration, full automation, and production deployment.*

4.4 RQ4: Trust determinants, failure modes, and quality expectations

Trust in LLM-migrated code was shaped by three related factors: expected failure modes, non-negotiable quality attributes, and provenance transparency/accountability. Respondents did not frame trust mainly as syntactic correctness or stylistic polish; instead, trust depended on whether LLM-generated migrations preserved behaviour, captured project-specific context, avoided security regressions, and remained reviewable.

At the workflow level, the leading issues were LLMs misunderstanding or missing project aspects, 45/61 (73.8%); missing company-internal knowledge, 34/61 (55.7%); ignored project requirements, 30/61 (49.2%); and inconsistent results across runs, 25/61 (41.0%). At the generated-code level, the leading concerns were incorrect or missing edge-case handling, 39/61 (63.9%); changed semantic behaviour, 36/61 (59.0%); unnecessary code, changes, or revisions, 33/61 (54.1%); and code not meeting project requirements, 25/61 (41.0%). Thus, dominant failure modes were semantic and context-sensitive rather than merely syntactic: respondents worried whether LLMs could preserve system behaviour, respect implicit requirements, and handle project-specific knowledge. The quality-expectation ratings reinforce this interpretation. The highest-rated requirements were semantic equivalence, rated important or very important by 90.2% of respondents, median = 5; absence of syntax errors, 86.9%, median = 5; and security, 83.6%, median = 5. Test cases were also highly rated, with 78.7% rating them important or very important. Thus, trust in LLM-migrated code depends on more than whether the output compiles. Respondents expected migrated code to preserve behaviour, remain secure, and be supported by verification evidence.

Security emerged as a strong trust boundary: it was both a highly rated quality expectation and the only engineering outcome with a net negative perceived impact from current LLMs. Respondents were optimistic about productivity, 86.9% positive, mean impact = 0.87, and developer experience, 67.2% positive, mean = 0.62, but more cautious about review process, code quality, maintainability, and bug detection. Security stood out: 36.1% reported a negative impact, versus 14.8% positive, yielding a mean impact of -0.21. This suggests selective optimism: practitioners expect LLMs to improve speed and developer experience, but not necessarily the assurance properties needed for production trust. Subgroup comparisons suggest that trust expectations varied by context. Internal respondents placed higher importance on security than external respondents, mean 4.72 vs. 3.97, Mann-Whitney $p = 0.003$, effect ≈ 0.41 ; external respondents placed higher importance on tests, mean 4.50 vs. 3.64, $p < 0.001$, effect ≈ 0.58 . These results indicate different emphasis in the CHECK24 and external samples: internal respondents rated security more highly, whereas external respondents rated test evidence more highly. The study cannot determine whether these differences arise from production exposure, organizational context, technology stacks, migration experience, or other group characteristics. These subgroup interpretations remain associative rather than causal.

Trust was strongly shaped by provenance disclosure. A repeated-measures comparison across four code-source conditions showed a large effect, Friedman $\chi^2 = 85.03$, $p < 0.001$, Kendall's $W = 0.50$. Explicit notice increased trust for both LLM- and human-written code: LLM-authored code rose from mean 2.70 without notice to 3.74 with notice, Wilcoxon $p = 5.42 \times 10^{-10}$; human-written code rose from 3.50 to 4.17, $p = 2.84 \times 10^{-9}$. Descriptively, LLM-written code with explicit notice was trusted more than human-written code without notice, suggesting that provenance transparency can partly offset source-based distrust. However, disclosure did not remove the source effect: even with notice, human-written code was trusted more than LLM-written code, 4.17 vs. 3.74, $p = 0.00135$. Thus, trust depended not only on human or LLM authorship, but also on traceability and explicit provenance;

notice improved reviewability and accountability, while LLM authorship retained a residual trust penalty. The internal–external split also appeared in perceived outcome impacts. External respondents were more optimistic than internal respondents about current LLMs’ impact on maintainability, $p < 0.001$, code quality, $p < 0.001$, bug detection, $p = 0.001$, and developer experience, $p = 0.015$. Internal respondents were more sceptical of these outcomes and rated expected security impact less favorably. Across several measured outcomes, the CHECK24 sample expressed more cautious perceptions than the external sample. These differences should be interpreted as company-associated sample differences rather than as effects of production experience alone.

One nuance is that respondents did not generally claim that LLM outputs are held to a higher explicit standard than human outputs: 35/61 (57.4%) said people are held to a higher standard, 17/61 (27.9%) said the standard is equal, and only 3/61 (4.9%) selected LLMs. This does not imply a lower practical bar for LLM-generated migration code; rather, it suggests different default accountability. Human-generated migration code is expected to carry stronger ownership and responsibility, whereas LLM-generated code is treated as draft material requiring additional review before it can be trusted.

Qualitative responses support this interpretation. Respondents described LLM output as useful starting material, but not something to accept without deeper checking. They emphasized verifying semantic equivalence, inspecting edge cases, accounting for project-specific knowledge, and ensuring that humans understand and own high-stakes production changes.

Summary of RQ4. *Trust depended on semantic preservation, project context, security, verification evidence, and provenance transparency. Practitioners valued productivity gains but treated security as the main risk area. Provenance disclosure helped, yet LLM output remained draft material requiring substantial human review.*

5 Discussion

Overall, the results frame LLM-assisted migration as assurance-gated: useful for acceleration, but acceptable only when outputs are reviewable, verifiable, traceable, and human-owned.

LLM-based Migration Challenges in Practices. A key implication of RQ1 is that software migration is driven by long-term quality and security preservation rather than cost reduction. Consequently, LLM assistance cannot be evaluated solely by generation efficiency; it must support fundamental assurance goals. While LLMs accelerate the workflow, they simultaneously introduce substantial verification overhead. This conditional productivity highlights a critical trade-off: the implementation effort saved by LLMs must ultimately outweigh the added assurance burden required for validation.

Our findings reveal systematic practitioner ambivalence, marked by a clear distinction between partial automation and full autonomy. Practitioners accepted localized LLM support with manual review but rejected full-project automation and direct production deployment, regardless of prior experience. This suggests that they view LLMs as tools rather than accountable agents. Accordingly, migration tools should avoid end-to-end autonomous designs and instead support bounded migration units, expose intermediate transformations, and preserve human control over acceptance decisions.

Scope exposes the limits of local code-generation evidence. The sharp decline in acceptability from single-file to entire-project migration indicates that repository-level migration introduces systemic risks, including cross-file dependencies and implicit logic, making local correctness insufficient for industrial trust. To address this gap, we propose a three-level evaluation framework: local evidence, based on compilation and testing; integration evidence, based on cross-component consistency; and assurance evidence, based on reviewability, security, and accountability. Future evaluations should assess LLM-assisted migration across all three tiers rather than relying solely on localized functional metrics.

Trust depends on assurance, provenance, and accountability. Results from RQ4 indicate that trust in LLM-migrated code relies on an assurance context of provenance disclosure, reviewability, and accountability, rather than a mere preference for human authors. While practitioners acknowledge LLMs' productivity benefits, their distinct concerns regarding security reveal a separation between productivity gains and production-level trust. Because LLM-generated code lacks the implicit accountability of human developers, mere provenance disclosure is insufficient. Consequently, industrial LLM-assisted workflows must tightly integrate assurance mechanisms, such as security reviews, static analysis, and human inspection, rather than treating them as optional post-processing steps.

6 Implications for stakeholders

Our findings suggest that LLM-assisted migration should be designed and adopted as an assurance-gated workflow, with different responsibilities.

Practitioners. Practitioners should treat LLM-assisted migration as a scope- and evidence-gated activity, not as end-to-end automation. LLMs are best suited for bounded tasks such as single-file migration, API adaptation, boilerplate transformation, and exploratory first drafts. Before production use, teams should require semantic validation, regression tests, integration checks, security review, provenance information, and a named human owner. This shifts the workflow from “generate and deploy” to “generate, inspect, validate, and own”.

Tool builders. Tool builders should design migration copilots rather than autonomous migration agents. Tools should decompose migrations into reviewable units, expose intermediate transformations, preserve links between prompts, outputs, and changed files, and highlight assumptions made during migration. Beyond code generation, tools should reduce assurance effort by integrating build checks, test support, static analysis, dependency checks, security scanning, and reviewer-oriented change summaries. The design goal should be inspectable and traceable migration support, not only higher generation accuracy.

Engineering managers and organizations. Engineering managers should assess LLM-assisted migration beyond speed, since reduced coding effort may be offset by review, validation, and accountability work. Organizations should define policies for acceptable scope, required evidence, provenance retention, reviewer responsibility, and production approval. For production-critical systems, LLM-assisted migration should follow controlled change management with rollback plans, auditability, security gates, and explicit human ownership.

Researchers. Researchers should evaluate LLM-assisted migration beyond snippet-level correctness or local translation accuracy. Industrial acceptance depends on semantic preservation, project context, cross-component consistency, security, reviewability, provenance,

and accountability. Future studies should include repository-level dependencies, realistic build/test environments, reviewer workload, assurance evidence, and production-readiness criteria. The key question is not only whether LLMs can migrate code, but what evidence makes their output acceptable in practice.

7 Threats to Validity

We discuss the main threats to validity and the steps taken to mitigate them, while clarifying the scope of inference for our results.

Construct validity. Key constructs such as migration experience, maintainability, trust, acceptability, helpfulness, and automation may have been interpreted differently depending on respondents' roles, organizations, and prior experience. We mitigated this by deriving the questionnaire from survey guidelines, combining predefined options with open-ended responses, and refining the instrument through formative practitioner feedback. Respondents also based their answers on heterogeneous experiences with different LLMs, model versions, coding assistants, interfaces, organizational policies, and usage workflows. We aggregate these experiences under the broader category of LLM-assisted migration and do not estimate model- or tool-specific perceptions. Consequently, the findings characterize perceptions across heterogeneous LLM experiences and should not be interpreted as applying uniformly to every model, assistant, or migration workflow.

Internal validity. The study relies on self-reported data, which may involve recall bias, social desirability, or inaccurate estimates of migration experience, LLM usage, review willingness, and production readiness. We mitigated this by probing related constructs from multiple angles: scope, production sensitivity, review confidence, automation preference, quality expectations, and failure modes. We also applied a two-step filter based on target-population membership and completion time. Responses below the five-minute threshold were manually checked before exclusion; one also showed straight-lining, and no internal CHECK24 response fell below the threshold.

External validity. The final sample is limited and not statistically representative. The internal sample provides context-rich industrial evidence but may reflect one company's engineering culture, technology stack, migration history, and organizational policies. The open survey broadens the sample but remains a convenience sample distributed through professional networks. Pooling the two samples increases the breadth of perspectives but is not methodologically neutral, because aggregate patterns may be influenced by the specific organizational context represented by the 25 internal respondents. We therefore interpret the combined results as patterns within two complementary samples rather than as estimates for migration practitioners generally, and use subgroup comparisons to expose rather than eliminate possible contextual differences.

Conclusion validity. The quantitative analyses are exploratory. The sample size limits statistical power, especially for subgroup comparisons and logistic regression, and internal-external differences may be confounded by organization, role, migration type, production exposure, or prior LLM experience. We therefore avoid causal claims, use non-parametric tests for ordinal data, apply correction for post-hoc comparisons where appropriate, and interpret statistical results as associative tendencies in this sample.

Qualitative, temporal, and scope limitations. Open-ended responses helped explain quantitative patterns, but coding remains interpretive; we mitigated this through independent initial coding by two authors, disagreement resolution, and a shared codebook. Because LLM capabilities and industrial tooling evolve rapidly, the findings are a snapshot of practitioner perceptions at data collection time.

8 Conclusion

This study examines practitioner perceptions of LLM-assisted software migration through a survey of 61 industry professionals. The survey results show that practitioners value LLMs for improving productivity in localized, routine tasks but reject full-project automation and autonomous production deployment. Trust in LLM-generated code depends on semantic preservation, security validation, and explicit human accountability, with security identified as the primary risk factor. We therefore characterize LLM-assisted migration as an “assurance-gated” adoption problem. Future migration tools should align with industry expectations by prioritizing bounded scopes, reviewable artifacts, automated security checks, and clear human ownership.

Data Availability Statement

The accompanying data is available at <https://doi.org/10.6084/m9.figshare.32414763>, includes anonymized survey responses and question-level graphs.

References

- 1 Alan Agresti. *Categorical Data Analysis*. John Wiley & Sons, Hoboken, NJ, 3 edition, 2013.
- 2 Aylton Almeida, Laerte Xavier, and Marco Tulio Valente. Automatic library migration using large language models: First results. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '24*, pages 427–433, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3674805.3690746.
- 3 Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006. doi:10.1191/1478088706qp063oa.
- 4 Virginia Braun and Victoria Clarke. *Thematic Analysis: A Practical Guide*. SAGE Publications, London, UK, 2021.
- 5 Sivajeet Chand, Melih Kilic, Roland Würsching, Sushant Kumar Pandey, and Alexander Pretschner. Automated extract method refactoring with open-source llms: A comparative study. In *2025 2nd IEEE/ACM International Conference on AI-powered Software (AIware)*, pages 113–122, 2025. doi:10.1109/AIware69974.2025.00020.
- 6 Sivajeet Chand, Kevin Nguyen, Peter Kuntz, and Alexander Pretschner. Leveraging llms for multi-file dsl code generation: An industrial case study. *arXiv preprint arXiv:2604.24678*, 2026. doi:10.48550/arXiv.2604.24678.
- 7 Sivajeet Chand, Alexander Pretschner, Steve Haupt, Derui Zhu, and Sushant Kumar Pandey. DepWareTrans: Dependency-aware incremental repository migration across co-executable languages. Accepted at the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE 2026), 2026. URL: <https://mediatum.ub.tum.de/1858822>.
- 8 Valerie Chen, Jasmyn He, Behnjamin Williams, Jason Valentino, and Ameet Talwalkar. Beyond the commit: Developer perspectives on productivity with ai coding assistants, 2026. doi:10.48550/arXiv.2602.03593.
- 9 Edward E. Cureton. Rank-biserial correlation. *Psychometrika*, 21(3):287–290, 1956. doi:10.1007/BF02289138.

- 10 Gabriele Dario De Siano, Anna Rita Fasolino, Giancarlo Sperli, and Andrea Vignali. Translating code with large language models and human-in-the-loop feedback. *Information and Software Technology*, 186:107785, 2025. doi:10.1016/j.infsof.2025.107785.
- 11 Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. doi:10.1080/01621459.1937.10503522.
- 12 Aymeric Fromherz and Jonathan Protzenko. Compiling c to safe rust, formalized, 2024. doi:10.48550/arXiv.2412.15042.
- 13 Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.
- 14 Jaemin Hong. Improving automatic c-to-rust translation with static analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 273–277, 2023. doi:10.1109/ICSE-Companion58688.2023.00074.
- 15 David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. John Wiley & Sons, Hoboken, NJ, 3 edition, 2013. doi:10.1002/9781118548387.
- 16 Ali Reza Ibrahimzada, Kaiyao Ke, Mrigank Pawagi, Muhammad Salman Abid, Rangeet Pan, Saurabh Sinha, and Reyhaneh Jabbarvand. Alphatrans: A neuro-symbolic compositional approach for repository-level code translation and validation. *Proceedings of the ACM on Software Engineering*, 2(FSE), 2025. doi:10.1145/3729379.
- 17 Maurice G. Kendall and Bernard Babington Smith. The problem of m rankings. *The Annals of Mathematical Statistics*, 10(3):275–287, 1939. doi:10.1214/aoms/1177732186.
- 18 Ze Shi Li, Nowshin Nawar Arony, Ahmed Musa Awon, Daniela Damian, and Bowen Xu. Ai tool use and adoption in software development by individuals and organizations: A grounded theory study, 2024. doi:10.48550/arXiv.2406.17325.
- 19 Jenny T. Liang, Chenyang Yang, and Brad A. Myers. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024*, pages 52:1–52:13. Association for Computing Machinery, 2024. doi:10.1145/3597503.3608128.
- 20 Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. Guidelines for conducting surveys in software engineering. Technical report, Lund University, Lund, Sweden, 2015. URL: <https://lup.lub.lu.se/record/5366801>.
- 21 Yang Luo, Richard Yu, Fajun Zhang, Ling Liang, and Yongqiang Xiong. Bridging gaps in llm code translation: Reducing errors with call graphs and bridged debuggers. In *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 2448–2449, 2024. doi:10.1145/3691620.3695322.
- 22 Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. doi:10.1214/aoms/1177730491.
- 23 Matias Martinez and Bruno Gois Mateus. Why Did Developers Migrate Android Applications From Java to Kotlin? . *IEEE Transactions on Software Engineering*, 48(11):4521–4534, November 2022. doi:10.1109/TSE.2021.3120367.
- 24 Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. Lost in translation: A study of bugs introduced by large language models while translating code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24*, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3597503.3639226.
- 25 Shari Lawrence Pfleeger and Barbara A. Kitchenham. Principles of survey research: Part 1: Turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26(6):16–18, 2001. doi:10.1145/505532.505535.

- 26 Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20601–20611. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/ed23fbf18c2cd35f8c7f8de44f85c08d-Paper.pdf.
- 27 Baptiste Roziere, Jie Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. Leveraging automated unit tests for unsupervised code translation. *arXiv preprint*, 2021. [arXiv:2110.06773](https://arxiv.org/abs/2110.06773).
- 28 Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*, pages 181–190, 2018. doi:10.1145/3183519.3183525.
- 29 Jefferson Seide Molléri, Kai Petersen, and Emilia Mendes. Survey guidelines in software engineering: An annotated review. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16*, pages 58:1–58:6, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2961111.2962619.
- 30 Jefferson Seide Molléri, Kai Petersen, and Emilia Mendes. An empirically evaluated checklist for surveys in software engineering. *Information and Software Technology*, 119:106240, 2020. doi:10.1016/j.infsof.2019.106240.
- 31 Manish Shetty, Naman Jain, Adwait Godbole, Sanjit A Seshia, and Koushik Sen. Syzygy: Dual code-test c to (safe) rust translation using llms and dynamic analysis. *arXiv preprint arXiv:2412.14234*, 2024. doi:10.48550/arXiv.2412.14234.
- 32 Trevor Stalnakier, Nathan Wintersgill, Oscar Chaparro, Laura A. Heymann, Massimiliano Di Penta, Daniel M. German, and Denys Poshyvanyk. Developer perspectives on licensing and copyright issues arising from generative ai for software development. *ACM Transactions on Software Engineering and Methodology*, 35(2):1–39, 2026. doi:10.1145/3743133.
- 33 Bo Wang, Ruishi Li, Mingkai Li, and Prateek Saxena. Transmap: Pinpointing mistakes in neural code translation. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023*, pages 999–1011, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3611643.3616322.
- 34 Chaozheng Wang, Junhao Hu, Cuiyun Gao, Yu Jin, Tao Xie, Hailiang Huang, Zhenyu Lei, and Yuetang Deng. How practitioners expect code completion? In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023*, pages 1294–1306. Association for Computing Machinery, 2023. doi:10.1145/3611643.3616280.
- 35 Yanli Wang, Yanlin Wang, Suiquan Wang, Daya Guo, Jiachi Chen, John Grundy, Xilin Liu, Yuchi Ma, Mingzhi Mao, Hongyu Zhang, and Zibin Zheng. Repotransbench: A real-world multilingual benchmark for repository-level code translation. *IEEE Transactions on Software Engineering*, 52(2):675–690, 2026. doi:10.1109/TSE.2025.3645056.
- 36 Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. Codetransocean: A comprehensive multilingual benchmark for code translation. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 5067–5089, 2023. doi:10.18653/V1/2023.FINDINGS-EMNLP.337.
- 37 Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, and Ge Li. Exploring and unleashing the power of large language models in automated code translation. *Proceedings of the ACM on Software Engineering*, 1(FSE), 2024. doi:10.1145/3660778.
- 38 Xin Yin, Chao Ni, Tien N Nguyen, Shaohua Wang, and Xiaohu Yang. Rectifier: Code translation with corrector via llms. *arXiv preprint arXiv:2407.07472*, 2024. doi:10.48550/arXiv.2407.07472.

- 39 Bingzhe Zhou, Xinying Wang, Shengbin Xu, Yuan Yao, Minxue Pan, Feng Xu, and Xiaoxing Ma. Hybrid api migration: A marriage of small api mapping models and large language models. In *Proceedings of the 14th Asia-Pacific Symposium on Internetware*, Internetware '23, pages 12–21, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3609437.3609466.
- 40 Celal Ziftci, Stoyan Nikolov, Anna Sjövall, Bo Kim, Daniele Codecasa, and Max Kim. Migrating code at scale with llms at google. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, FSE Companion '25, pages 162–173, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3696630.3728542.