A Repository-First Framework for Democratizing Chatbot Development

1st Rashan Boonsri
Department of Science and Technology
Rajamangala University of Technology Suvarnabhumi
Nonthaburi, Thailand
167480322002-st@rmutsb.ac.th

2nd Utharn Buranasaksee
Department of Science and Technology
Rajamangala University of Technology Suvarnabhumi
Nonthaburi, Thailand
utharn.b@rmutsb.ac.th

***Abstract*—Teams often require chatbots that integrate with business systems; however, existing options involve trade-offs: hosted platforms enable quick setup but impose vendor lock-in, whereas custom frameworks preserve control but exclude non-developers. This study evaluated whether combining a repository-first framework with an AI coding agent can lower development costs and time without reducing reliability or usability. In a within-subjects experiment of eight matched chatbot app microtasks with nine developers and 144 runs, the framework plus agent condition completed tasks 56 percent faster with an agent-to-framework ratio of 2.29 and 95 percent confidence interval from 1.40 to 3.63, and at 43 percent lower cost with an agent-to-framework ratio of 1.76 and 95 percent confidence interval from 1.12 to 2.58 compared to the agent-only condition. Reliability signals improved with fewer human interventions and comparable error rates, token efficiency, and latency. A separate 50-user study reported a System Usability Scale mean of 70.6 with a 95 percent confidence interval from 68.6 to 72.6, exceeding the common benchmark of 68. These results support repository-first, agent-assisted development as a practical approach to democratize chatbot creation while maintaining portability and code ownership.**

***Index Terms*—chatbot development, AI-assisted coding, low-code platforms, citizen developers, software frameworks, human-in-the-loop**

I. Introduction

Chatbots are widely used for customer support, operations, and information access [1], [2]. Current solutions fall into two categories: hosted platforms (e.g., Dialogflow, Lex, Watson) [3] that emphasize ease of use, and open frameworks (e.g., Rasa, Botkit) [4] that emphasize extensibility and control. Platforms accelerate initial development but raise vendor lock-in and portability concerns [5], whereas frameworks preserve ownership but demand programming expertise [4].

Low-code and model-driven approaches aim to close this gap by converting high-level specifications into working chatbots [6], [7]. Tools such as Xatkit [8], [9] consolidate domain-specific languages (DSLs) and metamodel practices [10]–[12]. Meanwhile, Large Language Models (LLMs) are increasingly integrated into chatbot stacks [2], prompting calls for more rigorous evaluation of reliability and task success through automated dialogue benchmarks [13] and meta-analyses of LLM-as-a-judge methods [14].

This paper introduces a repository-first chatbot framework paired with an AI coding agent. The framework enables non-developers to contribute high-level specifications while preserving code ownership and portability. We evaluated the approach through controlled experiments, comparing it with an agent-only baseline on developer time, cost, reliability, and usability.

II. Related Work

Chatbot development solutions face a fundamental trade-off: platforms provide accessibility but impose vendor lock-in, whereas frameworks preserve control but require programming expertise. Recent AI-assisted and low-code approaches have attempted to bridge this gap; however, systematic evaluations for non-technical users remain limited. We examine four dimensions: (a) platform versus framework trade-offs, (b) low-code and model-driven approaches, (c) domain-specific implementations and evaluations, and (d) positioning of AI-assisted frameworks.

A. Platforms vs. Frameworks

Comparisons show that hosted platforms emphasize rapid GUI-driven setup, whereas open frameworks emphasize code-level control, on-premise deployment, and maintainability [4]. Comparative studies highlight differences across tools [15], [16], practitioner perspectives [17], and broader reviews [18]. Platforms risk vendor lock-in, whereas frameworks demand developer expertise [19], [20]. Recent studies have explored narrowing this gap with open low-code initiatives and LLM-assisted modeling [21], [22].

B. Low-Code and Model-Driven Approaches

Low-code and model-driven engineering (MDE) raise abstraction using DSLs, metamodels, and code generation. Model-driven and low-code approaches use DSLs, metamodels, and code generation to build chatbots [6], [12], with exemplars such as Xatkit [8], [9], and surveys consolidating DSL practices [10], [11], [23]. LLM-based automation now blurs the boundaries of low-code and no-code programming [22], [24], although users still face new notations and template-bound

C. Sector Reviews and Evaluation

Surveys in domains such as tourism and business services reiterate the trade-offs among flexibility, maintainability, and vendor dependency [25], [26]. Recent evaluations span traditional benchmarks and business studies [26], [27], automated task-oriented dialogue with LLM agents [13], and meta-analyses of LLM-as-a-judge methods [14], [28], [29]. Controlled and field studies on AI coding assistants further highlight their impact on completion time and developer effort [30], [31].

D. Positioning

Two gaps remain: (i) platform convenience versus framework portability, and (ii) expanding non-developer participation without steep DSL learning curves or new lock-in risks [4]–[7], [19], [20]. We address both with a repository-first framework and an AI coding agent that compiles high-level specifications into versioned code artifacts [22], [24], [32].

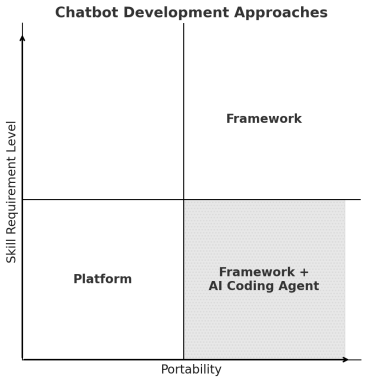


Fig. 1. Positioning of chatbot development approaches across portability and skill requirements. Our proposed framework with AI coding agent occupies the lower-right quadrant.

III. Proposed Framework

Objective. Enable *non-developers* to build, deploy, and maintain chatbots *without vendor lock-in* by pairing (i) a layered, repository-based framework with typed connectors, acceptance templates, and code scaffolds, and (ii) an *AI coding agent* that generates/edits codes from high-level task prompts.

A. Layered Architecture

We adopted a four-layer architecture with strict inward dependencies (i.e., a Clean Architecture).

Presentation layer: ASP.NET Core Minimal APIs and MVC controllers expose channel webhooks and admin UI for chatbot feature management, and SignalR provides real-time monitoring. Cross-cutting concerns include authentication, settings, security (signature checks, rate limits), and structured logging.

Application layer: Orchestration and business logic were implemented with CQRS using MediatR. Pipeline behaviors provide validation, performance metrics, and exception handling. A plugin mechanism registers new use-cases of the chatbot (feature) without modifying the core code.

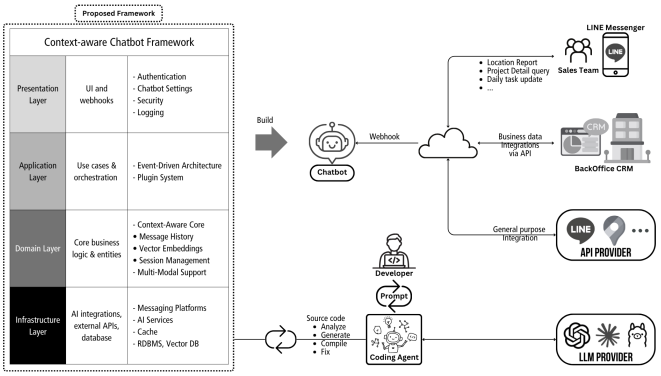


Fig. 2. Developing Chatbot framework with AI Coding agent

Domain layer: Enterprise rules and entities are independent of the infrastructure. A context-aware core maintains session state, message history, and optional *vector embeddings* for retrieval.

Infrastructure layer: Adapters for external systems and persistence PostgreSQL via EF Core pgvector for semantic search) Redis for session and distributed caching; messaging adapters LLM access via ModelHarbor. Google APIs, file handling and session cleanup.

B. AI Coding Agent Workflow

A domain expert fills a short playbook (intent, slots, and examples). The agent, backed by an LLM, analyzes the repository, generates handlers, routes, validators, and test stubs, runs smoke checks, and fixes regressions. Diffs and logs (transcripts and state deltas) were recorded. All artifacts remain plain Git files, preserving their ownership and portability.

C. End-User (Non-Developer) Flow

Business users extend chatbot functions through a lightweight review process.

1) Playbook. Select a predefined playbook, for example, Weekly Plan Inquiry, and provide fields (dates, descriptions, and locations).
2) Scaffold. The agent generates C# code and wiring (webhooks, connectors, and tests). The user reviews the diffs and accepts the changes.
3) Acceptance. Canonical prompts are executed; a reviewer (domain expert, team lead, QA) marks outcomes as pass/partial/fail against a checklist.
4) Deploy. Approved changes are committed and deployed, and new intents, validators, or connectors can be added later as plugins.

a) Reviewer role: In practice, the reviewer is not a full-time developer but acts as the quality gate. Depending on the context, this may be a domain expert who verifies correctness against business rules, a QA engineer who checks technical compliance, or a peer citizen developer. Over time, many of these checks may be automated, with human reviewers intervening only in exceptional cases.

D. Design Patterns and Extensibility

The framework applies common design patterns to maintain an organized and easily extensible code. Commands and queries were separated using the CQRS style with MediatR. Data access is handled through the Repository and Unit of Work patterns (EF Core). Cross-cutting concerns, such as validation, metrics, and error handling, are managed through pipeline behaviors. A Factory pattern is used to create processors dynamically when needed.

To connect new channels or data sources, developers only need to implement a small adapter interface that covers sending, receiving, schema mapping, and authentication processes. LLM providers and tool whitelists can be swapped without changing the rest of the system. Acceptance rules and slot validators can also be extended, thereby making the framework adaptable to different requirements.

E. Summary

The proposed framework contributes a repository-first development approach that supports the chatbot community by easing the *kickstart* process, it provides a ready-to-run toolkit that reduces setup challenges and accelerates early adoption. At the same time, the layered Clean Architecture offers an organized and structured foundation that remedies the limitations of traditional ad-hoc development methods, ensuring that projects are well-positioned for future feature expansion. The AI coding agent further lowers the skills barrier. non-developers can operate at the playbook level and still obtain auditable, versioned and vendor-neutral code artifacts.

From toolkit to evaluation: Having outlined the layered toolkit and agent workflow, we now examine how this approach supports real development tasks. Rather than replacing existing frameworks, our contribution demonstrates how a repository-first architecture, paired with AI coding support, can both streamline initial development and solidifies the base for long-term maintainability. To assess its practical value, we compare the Framework+Agent approach with an Agent-only baseline on matched LINE/CRM microtasks. Our evaluation considers developer efficiency (time and cost to completion), reliability signals (human interventions, errors, response efficiency, latency), and user outcomes (QA/acceptance and SUS). The next section details the experimental design used in this study.

IV. Methodology

A. Hypotheses and Evaluation Mapping

H1 – Chatbot accuracy (QA) - Measures: acceptance checklists, QA logs, Pass/Partial/Fail, FTR - Analysis: per-microtask accuracy, resolution rates, bootstrap

H2 – Developer efficiency and experience - Measures: completion time, cost, errors, interventions, token efficiency, latency, survey - Analysis: log-ratio, effect sizes, bootstrap, non-parametric, descriptive survey

H3 – End-user usability - Measures: SUS (50-user study) - Analysis: mean score, 95% CI, benchmark, qualitative feedback

B. Evaluation Overview

We evaluate how the proposed repository-first toolkit contributes to making chatbot development more accessible and efficient, while maintaining reliability and usability. To examine practical value, we ran controlled trials comparing Two conditions, Framework+Agent (F) and agent-only (A), were tested. We define *tasks* as three application-level scenarios related to LINE chatbot business use cases, each decomposed into a total of eight atomic *microtasks* (3+2+3 = 8). The microtasks (IDs 0101–0303) covered LINE API interfacing, message–data extraction, Google Sheets integration, and custom business logic, respectively. The unit of analysis is the *microtask*. Nine professional developers completed both conditions (within-subjects), yielding 9 replications per microtask in each condition (72 datapoints per condition, hence 144 in total).

While multiple outcomes were recorded (time, cost, reliability signals, and satisfaction), completion time served as the primary endpoint for power calculations because it was continuous, stable in pilot data, and directly linked to cost; other outcomes are analyzed in terms of effect sizes with Bootstrap confidence intervals.

Notation: We denote the two conditions as *A* (Agent-only) and *F* (Framework+Agent). For time and cost, we analyze the log ratio $\log(A/F)$. Values > 0 indicate an advantage for the Framework+Agent condition.

C. Design Constraints and Power

Because per-condition sessions were capped at 90 min, analyses were planned at microtask level with eight paired microtasks ($M = 8$). Repetitions within a microtask were used to estimate that microtask's per-condition means; inference is performed on the paired logratios across microtasks. Based on pilot variability, detecting a 20–25% time reduction with 80% power typically requires about 12–20 paired microtasks; with $M = 8$ we emphasize effect–size estimates (ratios of geometric means) with bootstrap 95% confidence intervals and corroborate with non-parametric tests (Wilcoxon/permutation), where appropriate.

Power / Sample Size (paired on microtasks): On the log scale, a 20–25% time reduction corresponds to mean log–ratios $\mu = \ln(1/0.80) = 0.223$ to $\ln(1/0.75) = 0.288$. With pilot log SD $\sigma \approx 0.35$, the standardized effect is $d = \mu/\sigma \approx 0.64$–$0.82$. A two–sided $\alpha = 0.05$ paired test requires approximately $M \approx \left(\frac{1.96+0.84}{d}\right)^2 = 12$–$20$ microtasks for 80% power.

In our design, $T = 8$ microtasks were attempted, each with 9 paired developer observation (one per condition). This provides 72 paired observations in total, However, the effective unit of analysis is the microtask, not the repetition. Thus, the study is under–powered for detecting a 20% reduction (≈ 44 power) and moderately powered for 25% ($\approx 65\%$). We therefore emphasize effect–size estimates with bootstrap 95% CIs rather than relying solely on p–values.

D. Participants

Nine professional developers participated in the study, all of whom were recruited internally.[1] On average, participants had 7.0 years of professional experience (SD = 2.05). Prior exposure to AI-assisted coding tools varied. five reported regular use (e.g., Copilot, ChatGPT) and four reported occasional use, ensuring representation across different levels of familiarity with the topic. The study was exempted from the institutional ethics review, and no production data were used.

E. Tasks

Eight matched microtasks were grouped into three application-level tasks that targeted the LINE chatbot. Each microtask had two isomorphic variants (A/B) that differed only in the identifiers and surface strings. An acceptance A checklist was defined for each microtask to verify its correctness.

- Task 1 (3 microtasks): Personalized Greeting *Acceptance:* Required integrating LINE Messaging services, involving: (1) Create a Processor for handling -Hello- message (2) Retrieve User's LINE ID from the message then call LINE API to get user's display name (3) prepare personalized contains user's display name and send a personalized message back to the user with their display name.
- Task 2 (2 microtasks): Location Echo with Greeting *Acceptance:* Required integrating LINE Messaging services, involving: (1) parsing location coordinates from LINE webhook payload (2) Prepare personalized contains user's display name, and location that were parsed from the user message and then sent a reply back to the user.
- Task 3 (3 microtasks): Google Sheets Nearby Places Lookup *Acceptance:* Required integrating LINE location services with Google Sheets API, involving: (1) parsing location coordinates from LINE webhook payload then querying Places sheet with haversine distance calculation, (2) Formatting results as LINE Flex Message with confirmation buttons, and (3) handling user selection to update the sheet. This represents a common business scenario where chatbot bridge messaging platforms with enterprise data sources.

F. Procedure

Participants received a 10–15 min standardized briefing and a prompt-library overview before starting the first condition. In *Framework+Agent*, the framework codebase and per-microtask prompt stubs were provided. In *Agent-only*, no framework code or orchestration was given; participants could use the same prompt stubs and were free to adapt or bring personal prompts. Each condition was limited to 90 min.

All prompts, responses, and tool invocations were automatically logged during sessions (see Section IV-J). To mitigate order effects, we applied an AB/BA counterbalanced design

[1]The event was organized as a friendly competition. Participants were not paid directly; instead, a gift voucher of about USD 60 was awarded to the fastest finisher.

Fig. 3. Left: AI generated code from Task 3. Right: LINE App chat message

with a Five-minute washout break between conditions. Each participant completed all eight microtasks (three in Task 1, two in Task 2, and three in Task 3), with isomorphic A/B variants that differ only in identifiers and surface strings.

G. Outcome Measures

a) Task success: Because the experimental deployment targeted a LINE webhook with Google Sheets integration, we verified success by manually auditing the chatbot's behavior against a pre-specified acceptance checklist (intent parsing, parameter extraction, data retrieval, and user-visible responses).Two raters independently reviewed each main task using a written checklist, and disagreements were resolved through discussion. We report completion as the proportion of runs that met all the criteria.

Primary: (1) Time-to-completion per subtask (minutes); (2) Cost-to-completion (USD); (3) Task success (pass/fail) via manual acceptance.

Secondary: (4) Human interventions/100 LLM calls (5) Errors/100 calls (6) Token efficiency (output/input ratio); (7) LLM calls and tool invocations; (8) Cost per minute;

H. Cost Calculation

Token usage was logged for both the input and output. Costs were derived by multiplying total tokens by the provider's published price per million tokens (USD) summed across all API calls in a trial. Reported cost ratios Thus, reflects the normalized end-to-end expenditure under each condition.

I. Technical Environment

OS: Windows 11 (build 26100). IDE: VS Code 1.103.0 with AI coding extension: ModelHarbor v3.25.10 (qwen3-235b-a22b-instruct-2507) in code mode with preconfigured

automated actions (file read/write, terminal commands, subtask execution). Qdrant for codebase indexing. External APIs: LINE Messaging and Google Sheets (keys provisioned for this study). Ngrok LINE Webhook Bridging.

J. Instrumentation and Logging

Start and stop times were automatically recorded for each prompt submission to the coding agent. In addition, we logged system prompts, user prompts, model responses, tool and command invocations, token counts, and process exit codes. All logging was performed through the built-in capabilities of VS Code IDE extension (ModelHarbor).

K. Developer Satisfaction Survey

Beyond objective performance measures, our study also examines how developers perceived the proposed toolkit. This dimension aligns with H2, which concerns developer satisfaction and overall experience when using the Framework+Agent approach. To assess this, we administered a post-task questionnaire after developers completed both conditions: Framework+Agent (F) and Agent-only (A). The survey captured perceived satisfaction, efficiency, and ease of navigation. Items used a 5-point Likert scale, with questions adapted from the standard usability and developer experience instruments. Results were analyzed for mean satisfaction, variance, and qualitative comments.

Note. While cost is organizationally relevant, our H2 developer-experience interpretation foregrounds time-to-completion and human interventions as the primary experiential outcome.

L. Statistical Analysis

We treated the task as a paired unit ($T = 8$). For each task t, we compute per–condition means (time, cost) across the balanced runs (9 per condition), then analyze the paired log–ratios $\Delta_t = \ln(\mathrm{A}_t) - \ln(\mathrm{F}_t)$. Time and cost effects are reported as ratios of geometric means $\exp(\bar{\Delta})$ with non-parametric bootstrap 95% CIs resampling tasks. Normality of Δ_t was assessed (Shapiro–Wilk); when appropriate we used a one–sample t–test on Δ_t (null $= 0$); otherwise a Wilcoxon signed–rank test. For completeness we report effect sizes as the mean log–ratio ($\bar{\Delta}$) and Cohen's d_z on Δ_t, and Cliff's δ for non-parametric cases.

Binary task success was adjudicated manually (pass or fail) by the experimenter. Because completion was near–ceiling, we primarily report proportions; where paired discordances existed at the run level, McNemar's test was used. Secondary reliability/efficiency outcomes (interventions/100, errors/100, response efficiency, USD/min, latency) are compared descriptively and, when tested, adjusted for multiplicity using Holm's method.

1) Data handling (H2: developer experiment): Trials with unrecoverable infrastructure failures (pre-specified) were excluded. All retained runs completed within the 90 min cap; therefore no right-censoring was applied. A trial counted as success only when the acceptance checklist (IV-E) for its task was fully satisfied; "partial" completions were treated as failures in the primary analysis of this study. Sensitivity checks that re-scored partials as successes did not alter the qualitative conclusions.

Balancing runs: All metrics were first aggregated at the participant×condition level. This avoids the unequal repetition of raw task runs. Where participants had missing values for a metric (e.g., no latency recorded or zero calls), they were excluded from the bootstrap sample of that metric. Analyses using all available runs versus participant-level aggregates produced the same direction and similar magnitudes of effects.

Rationale: Effects were computed at the participant level by aggregating all runs per condition, with non-parametric bootstrap resampling ($B = 10{,}000$). This avoids pseudo-replication from unequal task runs; sensitivity checks with all runs yielded the same effect direction and magnitude.

TABLE I
HEADLINE RESULTS ACROSS CONDITIONS

Metric	A/F ratio	95% CI	Interpretation
Completion time	2.44	[2.05, 3.02]	F faster overall
Total cost per task	1.88	[1.56, 2.15]	F cheaper overall
Human interventions/100	1.91	[1.45, 2.40]	F required fewer checks
Errors/100 calls	1.14	[0.70, 1.95]	No reliable difference
Token efficiency (out/in)	0.91	[0.81, 1.03]	Comparable
Cost per minute (USD/min)	0.85	[0.73, 0.99]	A slightly lower
Median feedback latency	1.07	[0.73, 1.45]	Comparable
SUS usability score	–	95% CI [68.6, 72.6]	Mean 70.6; 54% $\geq$ 68
QA accuracy	–	–	0.93
First-turn resolution	–	–	0.85

TABLE II
PER–TASK COMPARISON OF *Agent-only (A)* AND *Framework+Agent (F)*.

Task ID	Time A/F	Cost A/F	Favored
0101	1.83	2.02	F
0102	0.93	0.72	A
0103	2.26	2.22	F
0201	3.49	2.36	F
0202	6.33	3.52	F
0301	0.71	0.56	A
0302	3.23	2.31	F
0303	3.87	2.64	F

Note: Ratios A/F > 1 favor F.

V. EXPERIMENTAL RESULTS

We compared two conditions: Framework+Agent (F) and Agent-only (A). Task-level logs were aggregated per participant and condition, covering eight matched micro–tasks (Task IDs 0101–0303, corresponding to T1.1–T3.3 in our materials). From these, per–task means and global counters were computed.

A. Time and Cost

On average (Table I), F completed tasks faster than A (A/F ratio = 2.48, 95% CI [2.05, 3.02]) and reduced total cost (A/F ratio = 1.85, 95% CI [1.56, 2.16]). Per–task results (Table II; Figure 4) show that six of eight tasks favored F, with two low–complexity tasks (0102, 0301) favoring A.

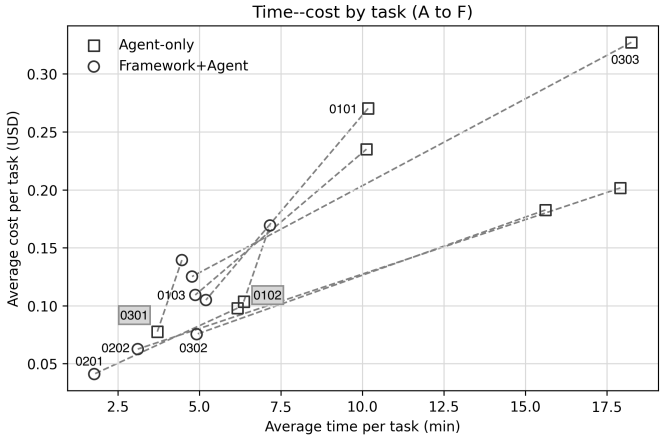


Fig. 4. Time–cost by task: the same task ID paired with dash line

B. Cost Decomposition

Although the framework's cost per–minute rate was modestly higher (A/F cost per minute = 0.85, 95% CI [0.73, 0.99]), tasks finished much faster under F (A/F time ratio = 2.44, 95% CI [2.05, 3.02]). The net effect was a lower end–to–end cost (A/F total cost ratio = 1.88, 95% CI [1.56, 2.15]) (Table I).

C. Reliability and Efficiency

Table I summarizes outcomes beyond time and cost. The framework+agent condition required significantly fewer human interventions per 100 calls (A/F ratio = 1.91, 95% CI [1.45, 2.40]), while error rates were statistically comparable (A/F = 1.14, 95% CI [0.70, 1.95]). Secondary efficiency signals showed only minor differences: token efficiency (output/input) was similar (A/F = 0.91, 95% CI [0.81, 1.03]). per-minute rates were modestly higher under F (A/F = 0.85, 95% CI [0.73, 0.99]), but this was offset by faster task completion, yielding lower total cost, and feedback latency showed no reliable differences (A/F = 1.07, 95% CI [0.73, 1.45]).

D. Developer Satisfaction Survey

To complement the quantitative time and cost measures, we administered a post-task survey after each developer completed both conditions. The survey assessed three dimensions of experience: satisfaction (S), perceived quality (Q) and perceived productivity (P). All items A 5-point Likert scale (1 = strongly disagree, 5 = strongly agree) was used. The overall composite score was calculated as the arithmetic mean of the three dimensions.

All three dimensions scored at or above 3.9 on average, indicating generally positive developer satisfaction in both conditions.

E. Summary and Interpretation

As summarized in Table I, the framework condition consistently improved the efficiency. It cut completion time and total cost despite a slightly higher per–minute rate, while reducing the need for human interventions and maintaining comparable

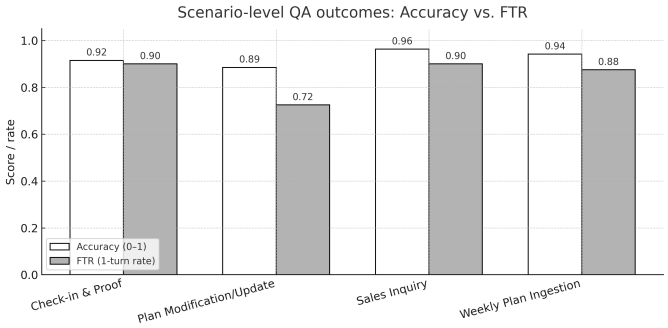


Fig. 5. H1: Normalized accuracy and First-turn resolution (FTR) by scenario

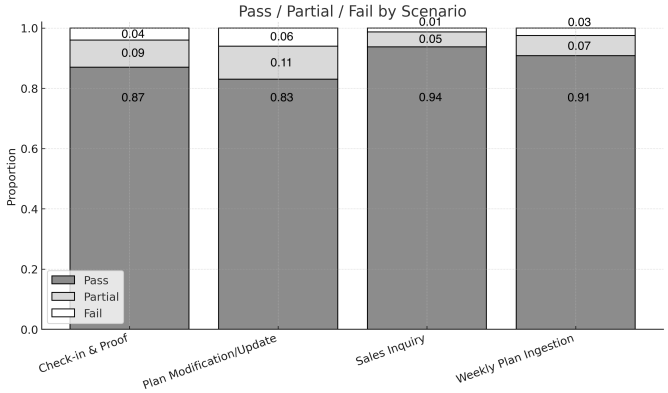


Fig. 6. H1: Pass / Partial / Fail by scenario.

error rates, latency, and token efficiency. Usability was strong overall (mean SUS 70.6; 54% $\geq$ 68), with additional QA and FTR outcomes that support reliability. Taken together, these results demonstrate that the proposed framework delivers faster, cheaper, and equally usable chatbot development compared with an agent-only baseline.

F. Takeaways

(1) The framework consistently reduces developer time, yielding lower end-to-end cost per task, despite a modest per-minute premium. (2) It reduces human supervision and feedback latency while keeping error incidence comparable and slightly improving response efficiency (tokens_out/tokens_in). (3) For very simple tasks with minimal state, Agent-only can match or beat the framework; the practical break-even is when $(t_A/t_F) > (r_F/r_A)$

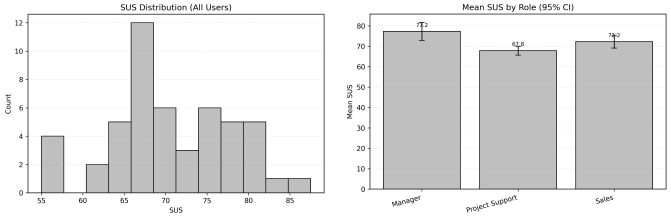
G. Hypothesis Outcomes

H1 (Chatbot accuracy; QA). Supported. Across scenarios, the normalized QA accuracy was 0.93 with acceptance 0.89 (Partial 0.08, Fail 0.03) in Table III. Resolution was efficient: all trials were completed within two turns ($FTR\leq 2 = 1.00$; mean 1.70 turns), and strict first–turn resolution was 0.25. Figure 5 summarizes accuracy and first–turn resolution; Figure 6 shows Pass/Partial/Fail, with per–scenario values in Table III. *Scope note*— These QA scenarios are domain–constrained (Figure 5), semi–structured sales workflows with stable intents

TABLE III
QA SUMMARY BY SCENARIO: PASS/PARTIAL/FAIL RATES AND NORMALIZED ACCURACY (0–1).

Scenario	Pass	Partial	Fail	Norm. Acc.
Check-in & Proof	0.87	0.09	0.04	0.92
Plan Modification/Update	0.83	0.11	0.06	0.89
Sales Inquiry	0.94	0.05	0.01	0.97
Weekly Plan Ingestion	0.91	0.07	0.03	0.95
Overall	0.89	0.08	0.03	**0.93**

Note. H1: Rates are proportions. Normalized accuracy = Pass + 0.5×Partial. FTR/1-turn rates in Fig. 5.



(a) SUS distribution ($N = 50$) (b) Mean SUS by role (95% CI)

Fig. 7. System Usability Scale (SUS) results. Left: overall distribution; Right: role-level means with 95% confidence intervals.

and slots. This yields high accuracy and acceptance; first–turn resolution is lower because many flows are explicitly two–step (request + confirmation), consistent with the observed mean of 1.70 turns.

H2 (Developer efficiency and experience). Supported. The framework+agent condition completed tasks 56% faster (A/F time ratio = 2.29, 95% CI [1.40, 3.63]) and at 43% lower cost (A/F = 1.76, 95% CI [1.12, 2.58]) than agent–only. It also required fewer human interventions (A/F = 1.91, 95% CI [1.45, 2.40]), while error rates were similar (A/F = 1.14, 95% CI [0.70, 1.95]). Secondary signals showed no major differences: token efficiency (A/F = 0.91, 95% CI [0.81, 1.03]) and feedback latency (A/F = 1.07, 95% CI [0.73, 1.45]) were comparable, and agent–only carried a modest per–minute cost advantage (A/F = 0.85, 95% CI [0.73, 0.99]). Post-task surveys (Section V-D) reported consistently high satisfaction and perceived efficiency of the system.

H3 (End–user usability; SUS). Supported. In a 50–user study, the SUS mean was 70.6 (95% CI [68.6, 75.9]), with 54% $\geq$ 68 (Figure 7a; Table IV). This exceeds the common acceptability benchmark of 68 and aligns with the qualitative feedback on workflow fit.

VI. THREATS TO VALIDITY

Internal: Learning or fatigue may bias the second condition in a crossover design; we mitigated this with AB/BA counterbalancing, isomorphic task variants, and scheduled washout breaks. Outcome adjudication risk was reduced through blinded raters when manual checks were required. Environmental drift (hardware, network, LLM service) was logged for each session.

Construct: The time and cost to completion are standard but do not fully capture code quality. We complemented them with secondary metrics (errors/100 calls, human interventions/100 calls, token efficiency) to better reflect the reliability and effort.

TABLE IV
SYSTEM USABILITY SCALE (SUS) RESULTS BY ROLE.

Role	N	Mean	Median	SD	% $\geq$ 68	95% CI
Manager	10	77.25	80.0	7.12	80	[72.84, 81.66]
Project Support	30	67.83	67.5	6.11	40	[65.65, 70.02]
Sales	10	72.25	72.5	5.06	70	[69.11, 75.39]
All users	50	70.6	70.0	7.1	54	[68.63, 72.57]

External: The tasks targeted LINE and Google Sheets in C#/.NET; generalization to other stacks may differ. As the participants were professional developers, the findings may not reflect novice workflows.

Conclusion: We report paired effect sizes, geometric mean ratios with bootstrap CIs, and pre-specified handling of the missing data. Multiple comparisons of the secondary outcomes were Holm-adjusted. Residual biases remain possible; all prompts, task variants, log schemas, and acceptance tests were released for replication.

Limitations: Because the session could only include eight paired tasks within the 90 min limit, the experiment had less statistical power than it would have had with a larger set of tasks. However, the observed effects were consistent and sizable (time A/F = 2.48, cost A/F = 1.85). We analyzed balanced repetitions (9 per microtask/condition), and the results were robust to the alternative use of all the available runs. Future work will broaden task diversity and apply mixed-effects models with task and developer random effects.

VII. CONCLUSION

This paper demonstrates that repository-first frameworks paired with AI coding agents can democratize chatbot development. Through controlled experiments (nine developers, 144 trials), we established that structured scaffolding reduces development time by 56% and costs by 43%, while maintaining comparable reliability metrics and quality outcomes compared to unstructured AI approaches. *Key insight*: AI assistance alone does not suffice for citizen developers. In the absence of architectural scaffolding, users required 1.91× more interventions for error recovery and greater assistance in completing complex integration tasks. The proposed repository-first architecture alleviates these challenges by producing versioned, portable artifacts that eliminate vendor lock-in and empower non-developers to contribute at the playbook level while preserving development ownership and control. The observed performance variations suggest enhancement opportunities through specification-driven developments. Integrating formal methods (user stories, decision tables) adapted for non-technical users could reduce ambiguity-related issues, improving consistency beyond current levels—a natural evolution in which specifications complement scaffolding. By identifying the complexity threshold at which frameworks become essential (beyond to 3-4 integrations), we provide practical guidance for task delegation strategies. Future work

includes (1) longitudinal studies with actual citizen developers, (2) specification template integration, (3) domain-specific playbooks, and (4) multi-organization validation. The open-source implementation enables the community to extend these findings further.

REFERENCES

[1] S. K. Dam, C. S. Hong, Y. Qiao, and C. Zhang, "A complete survey on LLM-based AI chatbots," *arXiv preprint arXiv:2406.16937*, 2024. [Online]. Available: https://arxiv.org/abs/2406.16937

[2] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024. [Online]. Available: https://arxiv.org/abs/2402.06196

[3] C. Ouaddi, L. Benaddi, A. Souha, and A. Jakimi, "A comparative and analysis study for recommending a chatbot development tool," in *2024 International Conference on Global Aeronautical Engineering and Satellite Technology (GAST)*. Marrakesh, Morocco: IEEE, Apr. 2024, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10520754/

[4] S. Perez-Soler, S. Juarez-Puerta, E. Guerra, and J. De Lara, "Choosing a Chatbot Development Tool," *IEEE Software*, vol. 38, no. 4, pp. 94–103, Jul. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9364349/

[5] IDC, "Open standards for the AI development ecosystem: Enabling integration, innovation, transparency, and freedom of choice," IDC White Paper, sponsored by Intel, Framingham, MA, Tech. Rep. US52685824, Dec. 2024. [Online]. Available: https://cdrdv2-public.intel.com/842312/US52685824.pdf

[6] D. Di Ruscio, D. Kolovos, J. De Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-code development and model-driven engineering: Two sides of the same coin?" *Software and Systems Modeling*, vol. 21, no. 2, pp. 437–446, Apr. 2022. [Online]. Available: https://link.springer.com/10.1007/s10270-021-00970-2

[7] A. A. Martinez-Garate, J. A. Aguilar-Calderon, C. Tripp-Barba, and A. Zaldívar-Colado, "Model-Driven Approaches for Conversational Agents Development: A Systematic Mapping Study," *IEEE Access*, vol. 11, pp. 73 088–73 103, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10177771/

[8] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Xatkit: A Multimodal Low-Code Chatbot Development Framework," *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8960373/

[9] G. Daniel and J. Cabot, "Applying Model-Driven Engineering to the Domain of Chatbots: The Xatkit Experience," *Science of Computer Programming*, vol. 232, p. 103032, 2023.

[10] A. Vahdati and R. Ramsin, "Model-driven methodology for developing chatbots based on microservice architecture," in *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2024)*. SCITEPRESS, 2024, pp. 247–254. [Online]. Available: https://www.scitepress.org/Papers/2024/124337/124337.pdf

[11] C. Ouaddi, L. Benaddi, E. M. Bouziane, A. Jakimi, A. Chehri, and R. Saadane, "A sketch of DSL and code generator for accelerating chatbot development," *Procedia Computer Science*, vol. 246, pp. 3585–3594, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924022142

[12] ——, "Dsl-driven approaches and metamodels for chatbot development: A systematic literature review," *Expert Systems*, vol. 42, p. e13787, 2025. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/exsy.13787

[13] A. Jain, P. Aggarwal, R. Sahay, C. Dong, and A. S. V. K. K. Saladi, "Autoeval-tod: Automated evaluation of task-oriented dialog systems," in *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), Long Papers*, 2025. [Online]. Available: https://aclanthology.org/2025.naacl-long.508.pdf

[14] J. Gu *et al.*, "A survey on LLM-as-a-judge," *arXiv preprint arXiv:2411.15594*, 2024. [Online]. Available: https://arxiv.org/abs/2411.15594

[15] H. Silkhi, B. Bakkas, and K. Housni, "Comparative analysis of rule-based chatbot development tools for education orientation: A rad approach," in *Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security (NISS '24)*. Meknes, Morocco: Association for Computing Machinery, 2024, pp. 1–7.

[16] B. Deshpande and M. Chandak, "Survey of Designing Tools for Chatbot Application," *International Journal of Health Sciences*, pp. 1403–1413, 2022.

[17] S. Käss, S. Strahringer, and M. Westner, "Practitioners' perceptions on the adoption of low code development platforms," *IEEE Access*, vol. 11, pp. 29 009–29 034, 2023.

[18] C.-M. Lin, A. Y. Huang, and S. J.-H. Yang, "A Review of AI-Driven Conversational Chatbots: Implementation Methodologies and Challenges (1999–2022)," *Sustainability*, vol. 15, no. 5, p. 4012, 2023.

[19] I. Alfonso, A. Conrardy, and J. Cabot, "Towards the interoperability of low-code platforms," *arXiv preprint arXiv:2412.05075*, 2024. [Online]. Available: https://arxiv.org/abs/2412.05075

[20] M. O. Ajimati, N. Carroll, and M. Maher, "Adoption of low-code and no-code development: A systematic literature review and future research agenda," *Journal of Systems and Software*, vol. 222, p. 112300, 2025.

[21] I. Alfonso, A. Conrardy, A. Sulejmani, A. Nirumand, F. Ul Haq, M. Gomez-Vazquez, J.-S. Sottet, and J. Cabot, "Building besser: An open-source low-code platform," *arXiv preprint arXiv:2405.13620*, 2024. [Online]. Available: https://arxiv.org/abs/2405.13620

[22] N. Hagel, N. Hili, and D. Schwab, "Turning low-code development platforms into true no-code with LLMs," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS Companion '24)*. Association for Computing Machinery, 2024, pp. 876–885.

[23] E. Martinez, L. Pfister, and F. Stauch, "Benefits and limitations of using low-code development to support digitalization in the construction industry," *Automation in Construction*, vol. 155, p. 105120, 2023.

[24] J. Sánchez Cuadrado, S. Pérez-Soler, E. Guerra, and J. de Lara, "Automating the development of task-oriented LLM-based chatbots," in *Proceedings of the 6th ACM Conference on Conversational User Interfaces (CUI '24)*. Luxembourg, Luxembourg: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3640794.3665538

[25] L. Benaddi, C. Ouaddi, A. Jakimi, and B. Ouchao, "A Systematic Review of Chatbots: Classification, Development, and Their Impact on Tourism," *IEEE Access*, vol. 12, pp. 78 799–78 810, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10542991/

[26] Y. Zhang, R. Y. K. Lau, J.-D. Xu, Y. Rao, and Y. Li, "Business Chatbots with Deep Learning Technologies: State-of-the-Art, Taxonomies, and Future Research Directions," *Artificial Intelligence Review*, 2024.

[27] A. Abdellatif, K. Badran, D. E. Costa, and E. Shihab, "A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 3087–3102, 2022.

[28] S. Raedler, L. Berardinelli, K. Winter, A. Rahimi, and S. Rinderle-Ma, "Bridging MDE and AI: A systematic review of domain-specific languages and model-driven practices in AI software systems engineering," *arXiv preprint arXiv:2307.04599*, 2023. [Online]. Available: https://arxiv.org/abs/2307.04599

[29] H. Li, W. Chen, Y. Wang *et al.*, "Llms-as-judges: A comprehensive survey on llm-based evaluation," *arXiv preprint arXiv:2412.05579*, 2024. [Online]. Available: https://arxiv.org/abs/2412.05579

[30] S. Peng and the GitHub Next Research Team, "The impact of ai on developer productivity: Evidence from github copilot," *arXiv preprint arXiv:2302.06590*, 2023. [Online]. Available: https://arxiv.org/abs/2302.06590

[31] K. Z. Cui and coauthors, "Evidence from a field experiment with github copilot," MIT GenAI Policy Network (PubPub) preprint, 2024. [Online]. Available: https://mit-genai.pubpub.org/pub/v5iixksv

[32] S. Muhammad, V. Prybutok, and V. Sinha, "Unlocking citizen developer potential: A systematic review and model for digital transformation," *Encyclopedia*, vol. 5, no. 1, p. 36, 2025. [Online]. Available: https://www.mdpi.com/2673-8392/5/1/36