

# What Challenges Do Developers Face in AI Agent Systems? An Empirical Study on Stack Overflow

Ali Asgari\*

a.asgari-2@tudelft.nl

Delft University of Technology  
Delft, The Netherlands

Pouria Derakhshanfar

pouria.derakhshanfar@jetbrains.com  
JetBrains Research  
Amsterdam, The Netherlands

Annibale Panichella

A.Panichella@tudelft.nl

Delft University of Technology  
Delft, The Netherlands

Mitchell Olsthoorn

M.J.G.Olsthoorn@tudelft.nl  
Delft University of Technology  
Delft, The Netherlands

## Abstract

AI Agents have rapidly gained popularity across research and industry as systems that extend large language models with additional capabilities to plan, use tools, remember, and act toward specific goals. Yet despite their promise, developers face persistent and often underexplored challenges when building, deploying, and maintaining these emerging systems. To identify these challenges, we study developer discussions on Stack Overflow, the world's largest Q&A site with 60 million questions & answers, and 30 million users. We construct a taxonomy of developer challenges through tag expansion and filtering, apply LDA-MALLET for topic modeling, and manually validate and label the resulting themes. Our analysis reveals seven major areas of recurring issues encompassing 77 distinct technical challenges related to runtime integration, dependency management, orchestration complexity, and evaluation reliability. We further quantify topic popularity and difficulty to identify which issues are most common and hardest to resolve, map the tools and programming languages used in agent development, and track their evolution from 2021 to 2025 in relation to major AI model and framework releases. Finally, we present the implications of our results, offering concrete guidance for practitioners, researchers, and educators on agents reliability and developer support.

## CCS Concepts

- Software and its engineering → Maintaining software; Software evolution.

## Keywords

AI Agents, Stack Overflow, Topic Modeling, Software Maintenance

\*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

*Software Engineering Venue 2026,*

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YYY/MM

<https://doi.org/XXXXXX.XXXXXXX>

## ACM Reference Format:

Ali Asgari, Annibale Panichella, Pouria Derakhshanfar, and Mitchell Olsthoorn. 2026. What Challenges Do Developers Face in AI Agent Systems? An Empirical Study on Stack Overflow. In *Proceedings of Submitted to a Software Engineering Venue (Software Engineering Venue 2026)*. ACM, New York, NY, USA, 12 pages. <https://doi.org/XXXXXX.XXXXXXX>

## 1 Introduction

*“We have been told 2025 is the year of AI Agents, but it is just the beginning.”* [17]. AI agents are the next step beyond generative AI and large language models (LLMs) due to their capabilities to perceive, reason, and act toward goals in their environment [24, 66, 73]. In software engineering (SE), agentic systems promise to reduce cost, time, and developer effort by automating workflows that span planning, tool use, memory, and interaction. An agent can be viewed as an intelligent entity that senses state, reasons over goals, and selects actions to maximize task performance [24, 73]. In SE contexts, an LLM-based agent typically comprises three modules: *perception, memory, and action* [73].

Despite their growing promise, the practical challenges of building and maintaining AI-agentic systems have not been systematically characterized. Recent studies have begun to examine why agent systems fail [21, 27, 28, 34, 43, 79], and Schneider et al.[66] outlines how agentic AI extends standard generative models. While this body of work highlights conceptual and architectural limitations, it primarily focuses on post-deployment behavior, narrow case studies, or system reports based on author experience rather than analyses of large, real-world developer data [29, 30, 64, 70]. In contrast, the day-to-day issues developers face (e.g., installing, integrating, debugging, or scaling agent frameworks) remain under-explored. A systematic, developer-grounded understanding of these in-practice challenges is still lacking. Addressing this gap is critical not only to improving the usability and reliability of agent frameworks, but also to guiding tool development, informing research priorities, and supporting developer education.

To address this gap, we analyze discussions from Stack Overflow, a large, developer-oriented Q&A platform (about 60 million questions & answers and around 30 million registered users). Stack Overflow data has been extensively used in empirical SE across domains such as Explainable AI [65], concurrency [5], GPU programming [75], Ruby programming [6], and deep learning [36]. Our goal is to systematically characterize the technical and practical

challenges developers face when building, deploying, and maintaining AI agents. We do so by formulating three research questions:

- RQ1:** *What topics do AI Agent developers ask about?*
- RQ2:** *Which topics are the most difficult and the most popular?*
- RQ3:** *What AI Agent technologies and programming languages are most used by developers?*

To answer RQ1, we mine Stack Overflow at scale, construct an agent-focused corpus via tag expansion and filtering, resulting in 3,191 unique questions and accepted answers. To cluster these, we apply LDA-MALLET for topic modeling [50], resulting in a taxonomy of seven major areas or recurring issues developers face with AI agents. We then draw a statistically representative sample of 343 questions for manual analysis, using Cochran's sample size formula with finite-population correction [74] and Neyman allocation [56] across topics. Through iterative manual coding and validation, we identify a taxonomy of 77 distinct technical challenges.

To assess which issues are most prevalent and hardest to resolve (RQ2), we compute topic-level popularity and difficulty scores [7, 71], finding that installation and dependency conflicts are frequent but relatively easy to resolve, while orchestration and RAG engineering are more difficult despite being less discussed. We observe a negative correlation between popularity and difficulty, suggesting that more visible issues tend to resolve faster, while RAG remains an outlier, being both complex and under-supported.

For RQ3, we profile the technology stacks developers use in agent systems. LangChain dominates as the orchestration framework (70.6% of posts), with OpenAI leading LLM usage. Python is the primary programming language (75.8%), followed by JavaScript and TypeScript. Temporal analysis shows that activity surged following the release of agentic primitives in mid-2023 and continued through 2024, aligning with major model and framework milestones.

Finally, we discuss the implications of our findings to guide future research on agent tooling and evaluation, supporting practitioners in prioritizing known pitfalls, and informing educators designing agent-oriented SE curricula. To enable replication and further exploration, we release all data, topic models, and coding artifacts in our replication package<sup>1</sup>.

## 2 Methodology

Although Stack Overflow provides structured content in the form of questions, answers, and metadata, it does not include explicit topic annotations relevant to AI Agent development. Therefore, to identify and analyze posts related to AI Agents, we first extract candidate posts and then categorize them according to their dominant themes. As illustrated in Figure 1, our methodology comprises seven steps to systematically identify and analyze Stack Overflow discussions related to AI Agent development. Each step incrementally refines the dataset, contributing to the reliability and interpretability of our findings. We describe each step in detail below.

**Step 1: StackExchange Data Explorer for Stack Overflow.** We use Stack Exchange Data Explorer<sup>2</sup>, a platform offering up-to-date and comprehensive data extracted from Stack Exchange 'data dumps' [65]. Among the various Stack Exchange communities, we

selected Stack Overflow as our target corpus for analysis. As discussed earlier, AI Agent development is an emerging and rapidly evolving topic. Preliminary inspection of other Stack Exchange sites (e.g., Artificial Intelligence, Data Science, Machine Learning) revealed that they currently contain insufficient or narrowly focused discussions about AI Agents. In contrast, Stack Overflow—hosting approximately 24 million questions, 36 million answers, and around 30 million registered users—offers a significantly larger and more diverse dataset<sup>3</sup> [4, 18, 31]. This scale and diversity provide a large corpus for investigating developer activities related to AI Agents and align with previous software-engineering mining studies that have successfully leveraged Stack Overflow data [4, 18, 31].

**Step 2: Identify AI Agent Tags.** To identify the most relevant AI Agent related tags, we followed established tag-expansion strategies from prior studies [2, 14, 63, 65]. We started querying with the *Agent* tag alone, which returns 1379 questions. To broaden coverage, we expanded our tag set by using the *Agent* co-occurring tags, with the focus to avoid adding noise questions to our data, because these new tags should help us find AI Agent posts only. We extracted all the *Agent* co-occurring tags from Agent-tagged posts, resulting in 1047 unique co-tags. Next, we applied two heuristic measures to filter and retain only meaningful tags: Tag Relevance Threshold (TRT) and Tag Significance Threshold (TST) [63, 65]. TRT measures how related a specific tag is to the Agent-tagged posts; TST measures of how prominent a specific tag is in the Agent-tagged posts. We calculate the TRT and TST values as follows [63, 65]:

$$\text{TRT}_{\text{tag}} = \frac{\# \text{ AI Agent posts}}{\# \text{ posts}}, \quad \text{TST}_{\text{tag}} = \frac{\# \text{ AI Agent posts}}{\# \text{ posts in the initial tag set}} \quad (1)$$

A tag was considered both relevant and significant if its corresponding TRT and TST exceeded predetermined thresholds. The first two authors, with expertise in LLM and AI Agent development (particularly for Software Engineering tasks), independently examined the tags under different TRT and TST numerical thresholds. For each tag, we inspected a randomly selected sample of posts to determine when the tags became less relevant to AI Agents; in addition, paying attention to the tag descriptions on Stack Overflow proved necessary and helpful. Ultimately, after testing TRT thresholds in the range (0.05, 0.10, 0.15, 0.20, 0.25, 0.30) and TST thresholds (0.001, 0.002, 0.005, 0.010, 0.015, 0.020, 0.30), we agreed on specific thresholds ( $\text{TRT} \geq 0.10$ ,  $\text{TST} \geq 0.002$ ), which align closely with previous studies [2, 63, 65, 71, 75]. This criterion retained 13 tags: *agent*, *agents*, *multi-agent*, *langgraph*, *langchain-agents*, *crewai*, *ms-autogen*, *phidata*, *openAI-agents*, *langchain*, *py-langchain*, *rag*, *retrievalqa*). We also included two additional tags on semantic grounds: *langchain-js* and *retrieval-augmented-generation*. Since *langchain* and *langchain-agents* already met the TRT-TST thresholds, including *langchain-js* ensured coverage of JavaScript-based agent frameworks. Similarly, *retrieval-augmented-generation* was retained to complement *rag*, as both represent the same conceptual paradigm in agent pipeline design.

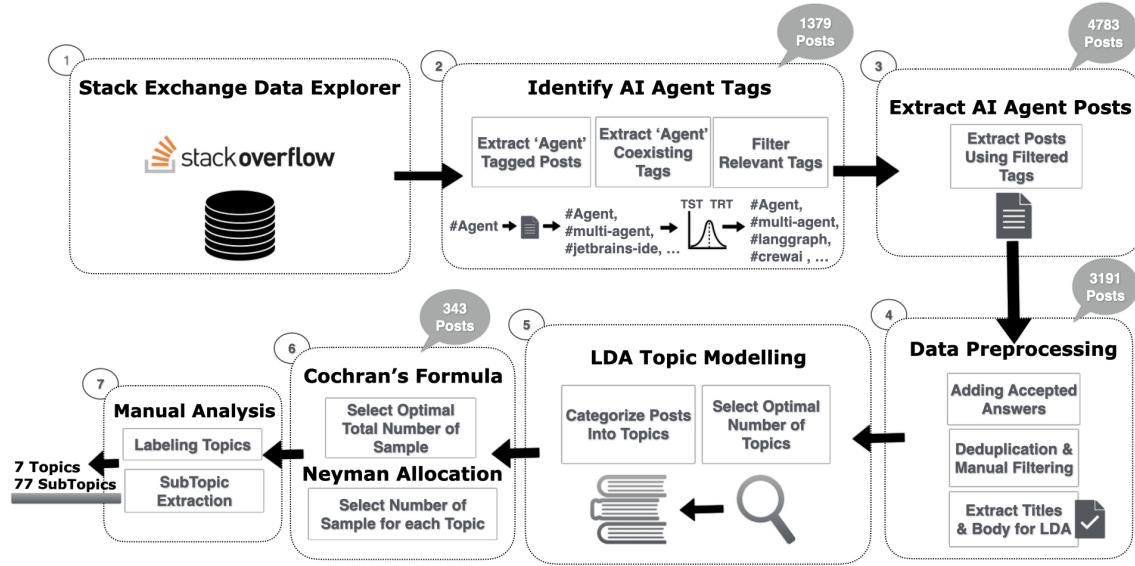
**Step 3: Extract AI Agent Posts.** After obtaining the AI Agent-related tag set, we used these tags to extract the posts that form our AI Agent dataset for this study. Following prior work [65, 69], we queried all Stack Overflow posts containing at least one of the

<sup>1</sup>Anonymous Zenodo replication package

<sup>2</sup>Stack Exchange Data Explorer. Available at <https://data.stackexchange.com>.

<sup>3</sup>List of Stack Exchange sites. <https://stackexchange.com/sites?view=list#users>.

Accessed 18 September 2026.



**Figure 1: Overview of the methodology of our study.**

selected tags and restricted the results to those with a PostTypeId of 1, thereby retaining only questions. This process yielded a dataset of 4,783 AI Agent posts along with their corresponding metadata.

**Step 4 : Data Pre-processing.** Before conducting topic modeling, we performed several text pre-processing steps to remove noise from the dataset. First, we removed duplicate posts, since searching each tag separately may return the same question multiple times (a post can have multiple tags). This eliminated 740 duplicates. Next, we restricted the dataset to questions posted after the first major AI impact on software development, the release of GitHub Copilot (2021-06-21), which removed 1,172 older posts. We then expanded the dataset by adding accepted answers from Stack Overflow using PostTypeId=2, yielding 530 accepted answers linked to 2,871 questions and resulting in 3,401 pairs of questions and answers. A final manual validation revealed irrelevant cases where the word ‘agent’ referred to AnyLogic or older DevOps concepts (e.g., Jenkins, Azure DevOps, GitLab runner, GitHub Actions). We excluded such posts through targeted keyword filtering, leaving a final dataset of 3,191 questions and accepted answers.

Following best practices from prior empirical studies on Stack Overflow mining [5, 16, 65], we further preprocessed the text by removing code blocks, HTML tags (e.g., `<p>`, `</p>`), URLs, and image tags [5, 16, 65]. We then applied a multi-stage natural language processing (NLP) pipeline to standardize and prepare the textual data. Using Gensim [60], we removed punctuation and tokenized the text. We further filtered out English stopwords from the NLTK corpus [48] and constructed bigrams using Gensim’s Phrases model to capture common multiword expressions (e.g., “language\_model” or “context\_window”). Finally, we applied spaCy [40] for lemmatization, reducing words to their canonical forms while retaining only the most semantically informative parts of speech: nouns, adjectives, verbs, and adverbs. The resulting corpus thus represents a

clean and linguistically normalized text dataset suitable for accurate and interpretable topic modeling.

**Step 5: LDA Topic Modeling.** Latent Dirichlet Allocation (LDA) is widely used for topic modeling in software repositories [39], including technical Q&A posts [16, 65] and issue reports. In this study, we employed the MALLET implementation of LDA [50], a method widely applied in software engineering research due to its higher coherence scores compared to the Gensim library [2, 46, 65]. The main challenge in using LDA lies in identifying the optimal number of topics ( $K$ ) for grouping the posts [59]. A high  $K$  produces highly specific topics, while a small  $K$  may result in overly generic topics. To address this, we examined  $K$  values ranging from 5 to 20 (step size of 1) and computed the coherence score for each configuration. The coherence metric measures the understandability of topics based on confirmation measures and has been shown to correlate strongly with human judgment [62]. LDA also leverages two crucial hyperparameters,  $\alpha$  (alpha) and  $\beta$  (beta), which govern the distribution of words across topics and the assignment of posts to topics [5, 9, 14, 63, 65, 71]. We adhered to conventional settings:  $\alpha = 50/K$ , where  $K$  is the number of topics, and  $\beta = 0.01$  [14]. These values are widely accepted in prior research and provide a robust benchmark for our experimental framework in topic modeling [65]. We found that the best coherence values were obtained for  $K = 7, 8, 6$ , and  $5$ . Among these,  $K = 7$  produced a coherence score of 0.462, which is the highest among all others  $K$  values. Although we inspected the four configurations, the separation of topics and the allocation of posts within them were clearer and easier to interpret when  $K = 7$ . The first two authors independently verified this by examining the top 20 posts per topic with the highest assignment confidence. Based on both the coherence analysis and qualitative validation, we selected  $K = 7$  as the optimal number of topics.

**Table 1: Neyman allocation by topic (Population  $N = 3191$ ;  $n_0 = 384.2$ ; FPC-adjusted  $n = 343$ ).**

Topic	Stratum Size $N_h$	Within SD $S_h$	Allocated $n_h$
1	390	0.109269	41
2	554	0.116195	62
3	309	0.110661	33
4	415	0.107397	43
5	667	0.115997	75
6	312	0.119681	36
7	544	0.101866	53
<b>Total</b>			<b>343</b>

**Step 6: Sampling.** Our dataset contains 3,191 question-answer records distributed across seven topics of varying sizes. To validate the LDA results, we apply sampling that accounts for both topic volume and the model’s confidence in assigning topics to records [12, 65], making manual analysis feasible [2, 5, 31, 71]. To this aim, we use Cochran’s formula [61, 74]:

$$n_0 = \frac{z^2 p(1-p)}{e^2}, \quad n = \frac{n_0}{1 + \frac{n_0-1}{N}} \quad (2)$$

with margin of error  $e = 0.05$  and confidence level 0.95 ( $z \approx 1.96$ ) and estimated proportion  $p = 0.5$ . This results in a sample size of  $n_0 = 384.2$ . Because the original formula assumes an infinite or very large population and our corpus is finite ( $N = 3191$ ), we applied the finite population correction (FPC; second part of Eq. 2) [25, 56, 57, 61], yielding an adjusted sample size of  $n = 343$ . We then used Neyman’s allocation (Eq. 3) to distribute the total sample across topic strata so that stratum samples sum to  $n$  [45, 56]:

$$n_h = n \cdot \frac{N_h S_h}{\sum_{j=1}^H N_j S_j} \quad (h = 1, \dots, H) \quad (3)$$

This allocation uses each stratum’s size and within-stratum variability; specifically, to favor higher-confidence exemplars while retaining diversity, we set  $S_h$  to the sample standard deviation of Topic\_Perc\_Contrib (a confidence-like measure) within stratum  $h$ , reported as *WithinSD* in Table 1. Within each stratum, we then drew a probability-proportional-to-size (PPS) sample with selection probabilities proportional to Topic\_Perc\_Contrib.

**Step 7: Manual Analysis.** Consistent with previous studies on qualitative topic interpretation [5, 7, 14, 46, 65], we employed a card-sorting approach to label and validate the topics generated by LDA. Card sorting provides a balance between subjective interpretability and systematic categorization [33, 65, 71]. The first two authors, with expertise in LLM and agent development and testing, independently evaluated the sampled questions of each topic along with the top keywords associated with them. The third author reviewed all stages conducted by the first two authors to ensure accuracy and consistency. After this review, the authors engaged in collaborative discussions that produced an initial naming of the topics. Agreement was substantial: with a Cohen’s Kappa of  $\kappa \approx 0.67$  and a percentage agreement of  $P_o = 71.4\%$  [51]. After resolving minor discrepancies through discussion, the authors reached consensus on the definitive names of the seven topics, each representing a distinct challenge area within AI Agent development. To gain deeper insight into these themes, we conducted a sub-topic analysis leveraging the Neyman-allocated samples (Table 1). The first two authors analyzed the sampled records for each topic (Table 1) and proposed candidate

sub-topics based on the initial naming step. They iteratively refined their proposals through discussion, eventually reaching agreement [65, 71]. After approximately ten rounds of deliberation, the team converged on a stable categorization, resulting in the identification and naming of 77 sub-topics.

**Popularity & Difficulty of Topics.** We investigated the popularity of AI Agent topics among developers using three widely adopted metrics from prior work [2, 7, 11, 63, 65, 71, 76]. The first is the average view count, reflecting how often posts are accessed. The second is the average score of posts, capturing the community’s perceived usefulness. The third is the average comment count, which reflects engagement and interaction around a topic [67, 76].

To assess topic difficulty, we applied two established measures [2, 11, 14, 16, 63, 65]. The first is the percentage of questions without accepted answers, denoted as the unaccepted answers percentage. The second is the median time (in hours) required for a question to receive an accepted answer. Longer delays indicate higher difficulty. Evaluating popularity and difficulty across multiple metrics can be complex; we used two fused metrics following prior research [7, 71].

**Fused Popularity.** For each of the seven AI Agent topics ( $K = 7$ ), we standardized three popularity metrics by dividing each value by its mean across all topics. This yields normalized measures  $\hat{V}_i$ ,  $\hat{S}_i$ , and  $\hat{C}_i$ , representing views, scores, and comment counts respectively. The fused popularity  $P_i$  for topic  $i$  is [7, 71]:

$$\hat{V}_i = \frac{V_i}{\frac{1}{K} \sum_{j=1}^K V_j}, \quad \hat{S}_i = \frac{S_i}{\frac{1}{K} \sum_{j=1}^K S_j}, \quad \hat{C}_i = \frac{C_i}{\frac{1}{K} \sum_{j=1}^K C_j} \quad (4)$$

$$P_i = \frac{\hat{V}_i + \hat{S}_i + \hat{C}_i}{3} \quad (5)$$

where  $V_i$ ,  $S_i$ , and  $C_i$  denote the average number of views, score (upvotes–downvotes), and comments per question in topic  $i$ ;  $\hat{V}_i$ ,  $\hat{S}_i$ , and  $\hat{C}_i$  are their normalized values relative to the global topic averages; and  $P_i$  is the fused popularity score for topic  $i$ , where higher values indicate greater visibility and engagement.

**Fused Difficulty.** Analogously, we standardized two difficulty measures across all topics and averaged them. The fused difficulty for topic  $i$ , denoted  $D_i$ , is expressed as [7, 71]:

$$\hat{P}_i = \frac{P_i}{\frac{1}{K} \sum_{j=1}^K P_j}, \quad \hat{T}_i = \frac{T_i}{\frac{1}{K} \sum_{j=1}^K T_j}, \quad D_i = \frac{\hat{P}_i + \hat{T}_i}{2} \quad (6)$$

where  $P_i$  is the percentage of questions in topic  $i$  with accepted answers (proxy for success rate),  $T_i$  is the median hours to receive an accepted answer (proxy for response time),  $\hat{P}_i$  and  $\hat{T}_i$  are the normalized values relative to their means across all  $K$  topics, and  $D_i$  is the fused difficulty score. Higher  $D_i$  values indicate lower success rates and slower answers.

Finally, we analyzed the correlation between popularity and difficulty using Kendall’s  $\tau$  correlation coefficient [42], which is more robust to outliers than the Spearman’s correlation [44] and does not require data to be normally distributed as with the Pearson correlation. Note that similar to previous studies [2, 7, 65, 71, 75], the temporal evolution of these metrics could not be studied as the Stack Overflow lacks sufficient time-series granularity (e.g., view count and related metrics are not consistent over time).

**Table 2: Topics and subtopics with within-topic percentage usage.**

Topic 1 – Operations (Runtime & Integration)	Topic 2 – Document Embeddings & Vector Stores	Topic 3 – Robustness, Reliability & Evaluation	Topic 4 – Orchestration
<i>Topic share: 12.2% of posts</i>	<i>Topic share: 17.3% of posts</i>	<i>Topic share: 9.7% of posts</i>	<i>Topic share: 13% of posts</i>
<b>Subtopic</b> %	<b>Subtopic</b> %	<b>Subtopic</b> %	<b>Subtopic</b> %
File I/O & Preprocessing Pipelines 21.95	Chunking & Document Modeling (JSON/HTML/CSV) 24.19	Environment Consistency & Reliability 21.21	Tool-Use Coordination Policies 23.26
Cloud/Agent Integrations 19.51	Performance(filters, timeouts, API drift) 17.74	Interface Contracts & Structured I/O 18.18	Observability 11.63
API/SDK Auth & Configuration 17.07	Persistence & Collections (namespaces, save/load) 14.52	Agent Instrumentation & Execution Traces 12.12	External Orchestrators & Services 9.30
Containerization & Build Systems 9.76	Multi-Index/Store Strategies 9.68	Tool/Memory Binding 12.12	State Modeling & Channels 9.30
CI/CD & Environment Management 7.32	Retrieval Quality & Tuning 8.06	Planning & Multi-Agent Algorithm Correctness 9.09	Streaming & Real-Time I/O 6.98
App/Framework Integration (Web/Backend) 7.32	Weaviate/Azure AI Search Configuration 8.06	Evaluation Schedules & Metrics 6.06	App Embedding & Endpoints 6.98
Local LLM Runtimes & Backends 4.88	Metadata & Filters (where/ODATA; args) 6.45	Training Stability & Rewards 6.06	Policy Gates & Governance 6.98
Networking, Connectivity & Timouts 4.88	Scores & Similarity (cosine/L2; scores; all res.) 6.45	Reproducibility & Checkpointing 3.03	Graph Composition & Subgraphs 4.65
Streaming & Concurrency 4.88	Pinecone Integration & Issues 3.23	Security/Compliance Controls 3.03	Workflow Testing & Simulation 4.65
GPU/CUDA & Performance Tuning 2.44	FAISS / Indexing Errors 1.61		Routing & Control Flow 4.65
			Multi-Agent Topologies 4.65
			State Isolation & Merging 2.33
			Human-in-the-Loop & Interrupts 2.33
			Timeouts, Retries & Cancellation 2.33
Topic 5 – Installation & Dependency Conflicts	Topic 6 – RAG Engineering	Topic 7 – Prompt & Output Engineering	
<i>Topic share: 20.9% of posts</i>	<i>Topic share: 9.8% of posts</i>	<i>Topic share: 17% of posts</i>	
<b>Subtopic</b> %	<b>Subtopic</b> %	<b>Subtopic</b> %	
LangChain/LlamaIndex Version Drift (API Churn) 31.88	Ingestion & Document Processing (PDF/XM- L/Images) 12.12	Prompt composition & context injection (dense vs. answer; context-only answers) 18.87	
Python/Pydantic/Typing Compatibility 14.49	Scaling, Concurrency & Throughput 12.12	Agents & tool/function calling (incl. output parsers) 13.21	
Third-Party SDK Surface Changes 14.49	Evaluation, Logging & Traceability (RAGAS; sources) 9.09	Memory prompts & context-window control (buffers/windows/summarization) 11.32	
Non-Python Platform/Library Incompatibility 7.25	Prompting & Query Strategy (multi-query, guardrails) 9.09	Prompt templating & variable injection 11.32	
Vendor SDK/Client Mismatch (OpenAI/Azure/- Groq/Ollama) 7.25	Semantic Caching & Memoization 9.09	Chat templates & role prompting (Ollama/Llama) + stop sequences 9.43	
OS/Binary Environment Crashes 5.80	Session State & Multi-tenant Memory 9.09	Determinism, sampling & output length control 7.55	
Vector Store Client→Server/API Mismatch 5.80	Tokenization, Budgets & Cost Control 9.09	LCEL composition & chaining patterns 7.55	
Data Encoding/Serialization Breakages 4.35	Architecture & Framework Choices 6.06	Structured outputs (JSON, schemas, regex) 3.77	
Missing Extras/Optional Deps 4.35	Metadata & Splitter Control 6.06		
Frontend Loader & Worker Versioning (pdf.js) 1.45	RAG for Classification / Structured Data 6.06		
Observability/Tracing Setup Issues 1.45	Structured Outputs & Schema-Aware RAG 6.06		
Transformers Pipeline Interface Changes 1.45	Temporal & Freshness-Aware RAG 3.03		
	Vector Stores & Index Ops 3.03		

### 3 Results

#### 3.1 RQ1: Taxonomy of Challenges

Table 2 presents the major topics and sub-topics discussed in AI Agent-related posts on Stack Overflow, as identified in Step 7 of our methodology. We uncovered seven primary topics covering 77 distinct sub-topics, reflecting a wide range of developer concerns. These topics are: (1) Operations (Runtime & Integration) (12.2%), (2) Document Embeddings & Vector Stores (17.3%), (3) Robustness, Reliability & Evaluation (9.7%), (4) Orchestration (13.0%), (5) Installation & Dependency Conflicts (20.9%), (6) Retrieval-Augmented Generation (RAG) Engineering (9.8%), and (7) Prompt & Output Engineering (17.0%). The 77 sub-topics form a taxonomy of fine-grained technical challenges, capturing the diversity and complexity of real-world issues developers face. Below, we explore each topic in detail and its top most common challenges/sub-topics.

**Topic 1: Operations (Runtime & Integration).** This topic centers on deploying AI Agents as reliable production services: packaging artifacts into container images, pinning native dependencies, selecting and configuring execution backends (CPU/GPU/accelerators), wiring networks (ports, DNS, proxies), setting up identity and access (keys, tokens), and defining service endpoints. It also covers data paths (loaders, temporary storage, object stores), streaming and concurrency control (generators, async, back-pressure), and promotion of builds through CI/CD with environment parity across development, staging, and production. Finally, it includes the basics

of observability and operations, structured logs, traces, metrics, health checks, along with configuration and secrets management, resource allocation, and safe rollouts and rollbacks so that changes are deployed predictably and runs are reproducible. The top sub-topic is *File I/O & Preprocessing Pipelines* (21.95%): handling uploads, loaders, and temporary files so agents process user data reliably, e.g., [Q76261321](#). The second challenge is *Cloud/Agent Integrations* (19.51%): diagnosing agent service health, networking posture, and connector status in managed environments, e.g., [Q71696353](#).

**Topic 2: Document Embeddings & Vector Stores.** For AI Agents, document embeddings and vector stores provide working memory across steps, tools, and sessions. They hold task context, prior tool outputs, and domain snippets that the agent can look up quickly, filter by metadata, and rank by similarity. Practical concerns include how to split and model documents for retrieval, how to persist and update collections safely, how to filter by source or policy, how to route queries, how to expose scores to the planner, and how to keep performance stable as data and schemas evolve. The most frequent subtopics are *Chunking & Document Modeling* (24.19%), *Performance*(17.74%). Chunking & Document Modeling: choosing sensible split strategies for JSON so the agent retrieves coherent units, e.g., [Q78015622](#). Performance & Ops: mitigating slow metadata filtering in Chroma during large ingest, e.g., [Q78505822](#).

**Topic 3: Robustness, Reliability & Evaluation.** Examines how AI Agents are evaluated and hardened for reliable behavior

under real operating conditions. It covers how teams design evaluation schedules and metrics, test robustness of plans and tool calls, ensure reproducibility across runs and environments, and maintain traceable execution. The focus is agent-centric: planners should produce correct decisions, tool invocations should satisfy strict schemas, and results should stand up to repeatable assessment rather than one-off demos. The top sub-topics are *Environment & Deployment Reliability* (21.21%), *Interface Contracts & Structured I/O* (18.18%). The former is related to diagnosing configuration and permission issues across dev, CI, and production so evaluations are repeatable, e.g., [Q76815315](#). The latter is related to enforcing JSON or schema-validated tool inputs and outputs so plans evaluate reliably and fail fast on contract errors, e.g., [Q79134666](#).

**Topic 4: Orchestration.** This topic covers the control plane that turns agent capabilities into end-to-end workflows. It includes how tasks are decomposed, routed, and synchronized, how state and messages move through channels, how tools are invoked under explicit coordination rules, and how external services are integrated into a single execution graph. We consider both single-agent pipelines and multi-agent collectives: single agents still require clear routing, state discipline, and failure handling, while multi-agent systems add role design, handoffs, arbitration, and shared context. The emphasis is on architectural choices that make planning, delegation, message exchange, and recovery explicit, observable, and repeatable. Within this topic, the top challenge is *Tool-Use Coordination Policies* (23.26%), which is related to configuring when and how agents invoke tools, including disabling or sequencing parallel use to avoid conflicts, e.g., [Q79332599](#). The second most-frequent challenge is *Observability* (11.63%), related to capturing execution traces and operational errors in complex runs to diagnose control issues and architecture-related deadlocks, e.g., [Q79363673](#).

**Topic 5: Installation & Dependency Conflicts.** This topic addresses the setup and dependency management as first-class parts of agent systems. Agent stacks span Python packages, native libraries, vendor SDKs, CLI tools, and sometimes JavaScript or JVM components; small version shifts can break structured tool I/O, memory backends, or orchestration code. Core practices include pinning and resolving transitive dependencies, aligning type systems and serialization layers, selecting compatible client SDKs for model providers, and keeping environment definitions reproducible across developer machines, CI, and production. The top sub-topic is *LangChain/LlamaIndex version drift (API churn)* (31.88%): APIs and module paths move, causing imports and agent wiring to fail, e.g., [Q77338572](#). The second is *Python ↔ Pydantic/typing incompatibility* (14.49%): upgrades to Pydantic or typing semantics invalidate validators and models used for structured tool calls, e.g., [Q78613825](#).

**Topic 6 : RAG Engineering.** This topic centers on engineering retrieval-augmented chatbots as agentic systems that plan queries, ground answers in retrieved evidence, and manage state across turns. It covers ingestion pipelines for PDFs, XML, and images, query strategies that balance recall and precision, scaling paths for concurrent users, evaluation and logging to verify grounding, and controls for tokens and cost. The emphasis is on turning retrieval into a dependable capability inside the agent loop so tool calls, memory, and responses align with the application’s constraints. The most frequent sub-topic is *Scaling, Concurrency & Throughput* (12.12%): sizing pipelines and parallelism for many users while

keeping retrieval and generation responsive, e.g., [Q78259888](#). The second is *Ingestion & Document Processing* (12.12%): extracting structured content from PDFs or XML so chunks preserve tables, figures, and captions the agent can cite, e.g., [Q79586123](#).

**Topic 7: Prompt & Output Engineering.** The topic covers how agents are guided to produce the right behavior and outputs using prompt design and control levers. It includes template construction and variable injection for plan and tool contexts, memory prompts that shape how much history enters the window, role and chat templates that steer turn-taking, and mechanisms that constrain outputs into expected formats or schemas. It also considers the interaction between prompting and tool or function calling, sampling, and output length controls that influence determinism, and chaining patterns that keep prompts modular and testable. The most frequent sub-topic is *Prompt composition & context injection* (condense vs. answer, context-only answers) (18.87%), *Agents & tool/function calling (incl. output parsers)* (13.21%). The former focuses on designing prompts that enforce context-only answers and choosing between condense versus answer strategies, e.g., [Q77227902](#). The latter is related to shaping prompts to trigger or suppress tool calls and align with tool schemas or parsers, e.g., [Q77362103](#).

**Findings (AI Agent Topics).** Developers focus on making agents run reliably (e.g., installing dependencies, wiring integrations, and managing orchestration) more than on evaluation or retrieval design. Frequent breakages from evolving SDKs and libraries (e.g., LangChain, Pydantic) drive an emphasis on environment pinning, schema validation, and reproducible builds.

### 3.2 RQ2: Popularity & Difficulty

**Topic Popularity.** Table 3 summarizes per-topic averages for views, comments, answers, and score. *Installation & Dependency Conflicts* attracts the highest traffic and engagement (views and answers), followed by *Prompt & Output Engineering* and *Document Embeddings & Vector Stores*. *Orchestration* receives fewer views and answers, despite being central to multi-agent architectures, suggesting these issues may be both niche and under-documented relative to setup and prompting concerns.

**Topic Difficulty.** Table 3 reports the share of questions without an accepted answer and the median hours to an accepted answer. *RAG Engineering* is slowest to resolution and often remains without an accepted answer, with *Document Embeddings & Vector Stores* and *Orchestration* also showing long time-to-accept. In contrast, *Installation & Dependency Conflicts* tends to resolve quickly and more frequently, consistent with issues that, while common, have clearer fixes once the specific incompatibility is identified.

Table 4 reports Kendall’s rank correlations between per-topic popularity and difficulty. The clearest signal is *Comments (avg)* vs. *Median hours to accepted*, which shows a large and statistically significant negative association ( $\tau=-0.714$ ,  $p\text{-value}=0.030$ ): questions that draw more discussion tend to reach an accepted answer sooner. *Answers (avg)* is also negatively related to both difficulty measures ( $\tau=-0.429$ ,  $p\text{-value}=0.239$ ), and *Views (avg)* trends in the same direction ( $\tau=-0.333$ ,  $p\text{-value}=0.381$ ); these point estimates are consistent with “popular  $\Rightarrow$  easier,” but they are not significant at conventional levels and are suggestive not definitive. In contrast,

**Table 3: Popularity and difficulty of AI Agent topics.**

Topic	Views (avg)	Comments (avg)	Answers (avg)	Score (avg)	No accepted ans. (share)	Median hrs. to accepted	Fused Popularity	Fused Difficulty
1. Operations (Runtime & Integration)	1759.70	1.082	0.915	0.848	0.827	22.52	0.97	0.79
2. Document Embeddings & Vector Stores	2426.09	0.626	0.965	1.265	0.805	65.38	1.02	1.33
3. Robustness, Reliability & Evaluation	1441.01	1.031	0.855	0.953	0.772	16.92	0.92	0.68
4. Orchestration	1259.11	0.768	0.734	0.905	0.884	46.93	0.78	1.14
5. Installation & Dependency Conflicts	3231.54	1.166	1.096	1.200	0.785	11.71	1.30	0.62
6. RAG Engineering	1677.71	0.678	0.746	1.286	0.884	87.44	0.89	1.66
7. Prompt & Output Engineering	2621.59	0.778	0.975	1.409	0.836	21.51	1.13	0.78
All topics (avg)	2059.54	0.876	0.898	1.124	0.828	38.92	1.00	1.00

**Table 4: Kendall's  $\tau$ , popularity & difficulty (cell:  $\tau [p]$ ).**

Pop/Diff metrics	No accepted answer	Median hours to accepted
Views (avg)	-0.333 [0.381]	-0.333 [0.381]
Comments (avg)	-0.333 [0.381]	-0.714 [0.030]
Answers (avg)	-0.429 [0.239]	-0.429 [0.239]
Score (avg)	0.143 [0.773]	0.143 [0.773]

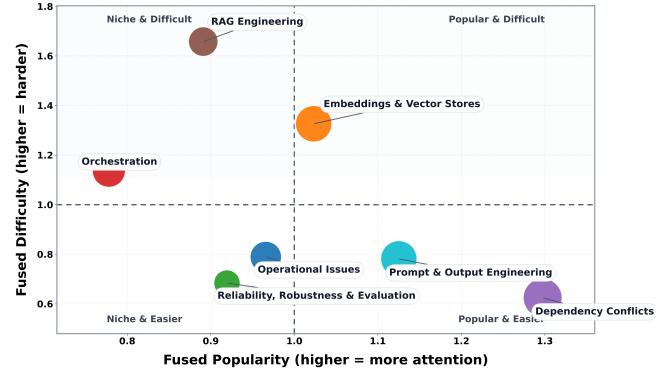
*Fused Popularity vs Fused Difficulty : -0.429 [0.239]*

Notes: Two-sided  $p$ -values. Negative  $\tau$  means higher popularity associated with lower difficulty.

*Score (avg)* has a weak, non-significant positive association with difficulty ( $\tau=0.143$ ,  $p$ -value=0.773). For completeness, note a strong correlation *within-popularity* not shown in the table—*Views (avg)* vs. *Answers (avg)*—which is significant ( $\tau=0.905$ ,  $p$ -value=0.003); this lies outside Table 4 because that table reports only pairs of popularity - difficulty. No other pairs reached statistical significance. Consistent with these pairwise patterns, the fused indices show a moderate negative relationship: *Fused Popularity vs Fused Difficulty* ( $\tau=-0.429$ ,  $p$ -value=0.239), indicating that, overall, more popular topics tend to be resolved faster and more often.

Fig. 2 visualizes the relation between *Fused Popularity* (x-axis) and *Fused Difficulty* (y-axis). The diagonal line marks equal popularity and difficulty; points below the line are easier than average relative to their popularity, while points above are harder than average. The size of each point reflects the number of posts in that topic, providing context on how much data underpins each observation. As we can observe, *Installation & Dependency Conflicts* is the most popular and also the easiest (pop. rank 1, diff. rank 7), while *Prompt & Output Engineering* is the second most popular with below-median difficulty (2, 5). *Document Embeddings & Vector Stores* is third in popularity but the second most difficult (3, 2), marking it as a high-demand yet thorny area. *RAG Engineering* is the most difficult despite being sixth in popularity (6, 1), and *Orchestration* is the least popular yet still hard (7, 3), suggesting concentrated challenges for fewer practitioners. *Operations (Runtime & Integration)* sits near the center (4, 4), and *Robustness, Reliability & Evaluation* is mid-low in popularity and relatively easier (5, 6).

**Findings (Popularity & Difficulty).** We observe a *moderate* popularity–difficulty tradeoff (popular topics tend to resolve faster), and we find a *significant* association between higher comment activity and *shorter* time-to-accept, suggesting discussion accelerates convergence. Two consistent outliers emerge: *RAG Engineering* remains the hardest despite middling popularity, and *Orchestration* is niche yet nontrivial (hard relative to its attention).

**Figure 2: Fused Popularity & Fused Difficulty.**

### 3.3 RQ3: AI Agent Technologies.

Table 5 summarizes the technologies mentioned by developers when building AI Agents, i.e., the libraries, services, and runtimes used to orchestrate tools, call models, retrieve and index context, compute embeddings, and evaluate outputs. Mentions cluster around *orchestration frameworks*, led overwhelmingly by *LangChain* (70.6%), with smaller but visible use of *LangGraph* (4.7%) and *CrewAI* (1.5%); *AutoGen* appears in 2.3% and is cross-listed because it is used both as an orchestration layer and a model/runtime interface.

Among *model APIs & runtimes*, *OpenAI* dominates (24.2%), followed by *Llama models/tooling* (5.0%), *Azure OpenAI* (3.8%), *Ollama* (3.8%), with *Google GenAI (Vertex/Gemini/PaLM)* (2.0%), *Anthropic* (0.9%), and performance-oriented runtimes such as *Groq* and *vLLM* (each 0.6%) appearing less often. For *retrieval & indexing*, *ChromaDB* is the most common store (9.6%), followed by *LlamaIndex* (4.1%), *FAISS* (3.8%), *Pinecone* (2.6%), and *Weaviate* (0.6%). Explicit *embedding Models* libraries are mentioned in a smaller share—*Hugging Face* (5.8%) and *Sentence-Transformers* (1.5%)—while *evaluation tooling* is rare; notably, *RAGAS* appears in 0.6% of posts and typically alongside RAG pipelines. Finally, *search services* account for modest use, including *Azure Cognitive Search* (2.0%) and *Elasticsearch/OpenSearch* (0.9%). Percentages are computed over all posts ( $N=343$ ); because posts can mention multiple technologies, category totals may exceed 100%.

**Programming Languages.** Figure 6 shows that Python overwhelmingly anchors AI Agent development, reflecting its rich ecosystem (orchestration frameworks, retrieval tooling, evaluation, and quick prototyping). Smaller but notable footprints come from *TypeScript/JavaScript* (front-ends, Node runtimes, *LangChain.js*), *Java* (enterprise/back-end integration), and *SQL* (data access and

**Table 5: AI Agent Technologies used on Stack Overflow.**

Category	Technology	Share (%)
Orchestration frameworks	LangChain	70.6
	LangGraph	4.7
	CrewAI	1.5
	AutoGen <sup>†</sup>	2.3
Model APIs & runtimes	OpenAI	24.2
	Llama models/tooling	5.0
	Ollama	3.8
	Azure OpenAI	3.8
	Google GenAI (Vertex/Gemini/PaLM)	2.0
	AutoGen <sup>†</sup>	2.3
Retrieval & Indexing	Groq	0.6
	vLLM	0.6
	ChromaDB	9.6
	Llamaindex	4.1
	FAISS	3.8
Embedding Models	Pinecone	2.6
	Weaviate	0.6
	Hugging Face	5.8
Evaluation	Sentence-Transformers	1.5
	RAGAS	0.6
Search services	Azure Cognitive Search	2.0
	Elasticsearch/OpenSearch	0.9
Other	Streamlit	2.9
	Neo4j	1.2
	Anthropic	0.9
	Flowise	0.6
	Azure AI Inference	0.3

<sup>†</sup>Cross-listed: AutoGen appears under both Orchestration and Model APIs because it is used as an agentic framework and as a model/runtime interface.

**Table 6: AI Agent Programming Languages.**

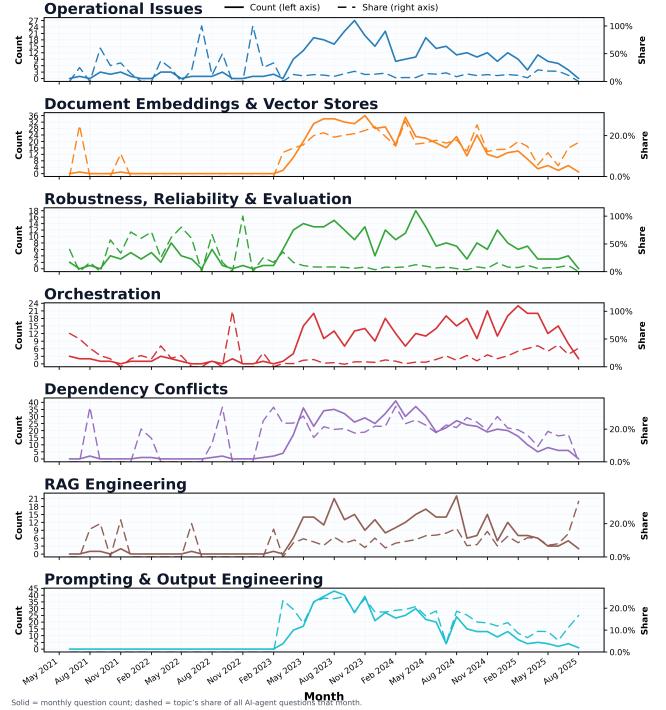
Language	%	Language	%
Python	75.8	Java	3.2
N/A	12.2	SQL	2.6
TypeScript	5.5	C#	0.3
JavaScript	5.0	C++	0.3

RAG pipelines). A visible N/A slice captures posts focused on concepts or provider APIs where the language is not specified. Note that the chart is multilabel; many posts mention more than one language, so percentages need not sum to 100%; the pattern nonetheless makes clear that Python dominates while web stack and enterprise languages play complementary roles.

**Findings (Technology Share).** AI Agent development is overwhelmingly *LangChain* + *OpenAI*-based, emphasizing quick orchestration over evaluation or measurement. Open-source and high-performance runtime appear selectively, while retrieval has consolidated around common vector stores. The ecosystem thus favors rapid composition and experimentation, with evaluation tooling still emerging.

## 4 Discussion

**Evolution of AI Agent Topics.** Table 5 defined what we mean by “AI Agent Technologies” (orchestration, model APIs/runtimes, retrieval/indexing, embeddings, evaluation, search, and other) and shows their shares; The time series in Fig. 4 begins in June 2021 (GitHub Copilot’s launch), stays relatively quiet through 2022, then surges in Q2 2023. This acceleration aligns with early RAG and agent primitives (e.g., vector stores, retrieval tooling), followed by OpenAI’s function calling (June 2023), the Llama family for

**Figure 3: AI Agent Topics Evolution.**

OSS/local models, and—critically—orchestration layers enabling robust multi-agent patterns (e.g., AutoGen and Assistants API in late 2023, LangGraph in early 2024). *LangChain*’s standardized prompting/tools/retrievers likely amplified this adoption wave. *OpenAI* remains the most referenced hosted model stack, while *Hugging Face* and *Llama* support local deployments; *LangGraph* appears in more advanced agent threads. Activity tapers after *GPT-4o* (May 2024) and early *GPT-5* news (2025), though this may reflect incomplete 2025 data. Overall, milestone cadence and usage trends align, suggestive of influence, though we avoid causal claims.

Figure 3 shows how each topic’s monthly share tracks major tool releases. *Installation & Dependency Conflicts* leads the late-2023 surge and pulses again in early 2024, consistent with SDK churn around RAG stacks and APIs (e.g., Chroma/LlamaIndex in spring–summer 2023, OpenAI functions in June, Assistants in November). *Prompt & Output Engineering* climbs from mid-2023, with spikes into early 2024, mirroring the shift to tool-calling schemas and multi-step controllers. *Document Embeddings & Vector Stores* ramps in mid/late 2023 (peaking in early autumn), then stabilizes through 2024–2025 as retrieval practices stabilize. *Orchestration* holds steady but ticks up post-graph-style frameworks (e.g., LangGraph in Jan 2024) and again into 2025. *RAG Engineering* shows a broad wave from mid-2023 to early 2024, especially late winter—then cools as templates mature. *Robustness, Reliability & Evaluation* appears in bursts in 2023, smoothing in 2024 with pockets citing evaluation tooling (e.g., RAGAS). Finally, *Operations (Runtime & Integration)* offers the long baseline from 2021, steps up in 2023, and recurs through 2024, reflecting the steady integration work trailing new capabilities.

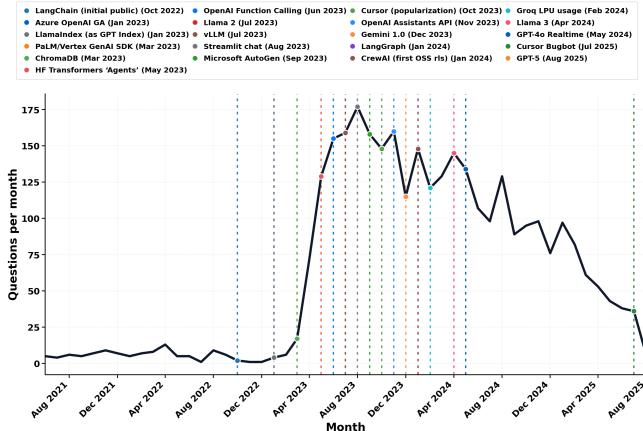


Figure 4: AI Agent Discussions Over Time.

Table 7: AI Agents vs. five prior Stack Overflow domains.

Metric	AI Chatbot Agents	Mobile Security	Big Data	IoT
<b>(Popularity)</b>				
Avg. View	2,240.1	512.4	2,300.0	2,461.1
Avg. Score	1.16	0.7	2.1	2.7
Avg. AnswerCount	0.93	1.0	1.5	1.6
<b>(Difficulty)</b>				
% w/o Accepted Ans.	82.6%	67.7%	52.0%	48.2%
Median Hrs. to Accepted	23.4	14.8	0.7	0.9
Notes: Prior-domain values reported from earlier SO studies [2, 5, 14, 63, 76].				

**Comparing to Other Domains.** As shown in Table 7—and in line with prior Stack Overflow studies on chatbots, mobile, security, big data, and IoT [2, 5, 14, 63, 76]—AI Agents stand out for difficulty rather than ease: they exhibit by far the highest share of questions without an accepted answer and the longest median time-to-accept, suggesting that solutions are harder to pin down and verification takes longer. Popularity-wise, AI Agents attract substantial attention: average views are on par with established domains (below security/mobile but well above chatbots and IoT), and average score is mid-range (higher than chatbots/IoT, comparable to big data, lower than security/mobile). At the same time, the average number of answers per question is lower than most comparison domains, reinforcing the picture of a rapidly growing but technically challenging space where community consensus forms more slowly. Overall, despite being comparatively new, AI Agents already command competitive engagement while posing above-average resolution difficulty.

## 5 Related Work

**Stack Overflow vs. ChatGPT.** Helcic and Santos [38] reported that the introduction of ChatGPT significantly influenced Stack Overflow activity—leading to longer and more complex questions and answers, as well as increased code length and question difficulty. They conclude that ChatGPT has effectively raised the overall bar for content complexity and challenge on the platform. Kabir et al. [41] found that participants in their study preferred human-generated Stack Overflow answers in 65.18% of cases. Similarly, Liu et al. [47] found that, in debugging tasks, ChatGPT performed

worse than Stack Overflow, and users interacting with ChatGPT took longer between their first execution.

**Stack Overflow Data & LDA Topic Modeling.** Previous work has mined Stack Overflow at multiple levels: from user commenting behavior and community dynamics [77], to downstream effects such as the impact of Stack Overflow-sourced code on mobile apps [1] and automated generation of code comments from snippets [3]. In parallel, researchers have used topic modeling on unstructured software repository data to systematically surface developer pain points [19], and applied LDA to Stack Overflow itself to map what developers discuss at scale [16]. Continuing this mindset, domain-specific studies have applied similar methods to particular ecosystems: Rosen et al. [63] used topic modeling on Stack Overflow to categorize mobile-development challenges (six topics); Uddin et al. [71] analyzed IoT questions with an empirical, topic-driven approach; Abdellatif et al. [2] studied chatbot development, organizing five categories across twelve topics; Openja et al. [58] examined modern release engineering practices; Elshan et al. [31] unveiled challenges in low-code software development (ten topics); Haque et al. [37] documented Docker-related issues; and Wang et al. [72] studied AutoML from a software-engineering perspective. AI and machine learning for software engineering also have been explored within development practices and data-driven insights at scale [10]; Sayyadnejad et al. [65] also identified ten topics about challenges of building explainable AI systems. Complementary studies have manually classified themes in Q&A platforms—e.g., MATLAB user issues [54], technical debt discussions [8], and code smells/anti-patterns in practice [69]. Building on two established pathways—automated topic modeling (e.g., LDA) and manual qualitative analysis—prior work has demonstrated that both can provide rich taxonomies of developer challenges. For example, [23] manually derived 25 topics and 72 subtopics related to the implementation of deep-learning-based software. In the same spirit, our study deliberately combines a community-standard validated approach for Stack Overflow data (LDA-Mallet [50]) [2, 46, 65], with a targeted manual analysis to refine subtopics and capture nuance—a strategy aligned with standard mixed-method empirical studies such as [49, 65, 71].

**AI Agents.** Cemri et al. [21] derive a data-driven taxonomy of multi-agent LLM failures—*specification, inter-agent misalignment, task verification*—aligned to pre/execution/post stages via manual coding of 200+ end-to-end traces. Different studies focus on distinct AI Agent challenges, including Security [27, 28, 79], Ethics & governance [34, 43], Foundations & effectiveness of multi-agent ai systems (MAS) [29, 30, 64, 70], and Design & applications [21, 53, 68]. And also Schneider et al. [66] in a conceptual survey clarify how Agentic AI extends GenAI (reasoning, tools, memory, interaction), formalize agent specification, and catalog open challenges.

**Security.** Deng et al. [28] survey the agent threat surface (intra-execution, agent-env, agent–agent, memory) and defenses (debate, RAG, constraints, post-correction, tool-use auditing). Zou et al. [79] show near-universal, transferable prompt-injection/policy-violation attacks in large-scale red teaming, exposing defense gaps. Witt et al. [27] argue for *multi-agent security* as a distinct field, outlining open problems (collusion, attribution, cascade dynamics) and a research agenda. **Ethics & governance.** Gabriel et al. [34] call for

new ethics for autonomous agents—safety, alignment, and social-coordination risks as autonomy grows. Kolt et al. [43] frame governance as a principal–agent problem and advocate transparency, incentive design, monitoring, and accountability across developers/deployers/users. *Foundations & effectiveness of MAS*. Tian et al. [70] formalize multi-agent AI, showing gains from task reallocation and diverse agents, but error amplification under overlap/misalignment; emphasize feedback that reshapes topology. Du et al. [30] survey multi-agent deep reinforcement learning, stressing non-stationarity, partial observability, scalability, and credit assignment. Du et al. [29] review agent–agent communication, distilling five pillars (scalability, security, real-time, performance, manageability) and standardization paths. *Design & applications*. Mo et al. [53] study an interactive refactoring agent, reporting decision strategies and collaboration principles. Shetty et al. [68] sketch requirements and a prototype (AIOpsLab) for a standardized framework to build/test/compare cloud-ops agents.

## 6 Implications

Our findings provide actionable insights for developers, researchers, and educators involved in building and teaching AI agent systems. Beyond identifying challenges, we translate these into concrete guidance: what to prioritize in practice, where to focus research efforts, and how to align educational content with real-world needs.

**For Practitioners.** Practitioners should prioritize stability early in the development cycle. Many issues stem from installation and dependency conflicts, which can be mitigated by pinning versions, maintaining lockfiles, using containers, and adding CI checks for API or import breakage across key libraries such as LangChain, LlamaIndex, and provider SDKs. Retrieval-augmented generation (RAG) should be approached as an engineering discipline, not a plug-and-play feature—investing in document modeling, retrieval evaluation (e.g., grounding and attribution), query strategies, and observability is critical before scaling usage. Although orchestration generates fewer questions, it remains a complex layer. Developers should prefer graph-based orchestration (e.g., LangGraph), enforce schema validation for tool calls, and implement trace logging. Multi-agent workflows (e.g., CrewAI) should be introduced only when roles and state transitions are well defined. Given the dominance of OpenAI, teams should maintain abstraction layers to enable quick pivots to Azure, Llama (via vLLM/Ollama), or Groq for cost, latency, or policy resilience. Minimal instrumentation can surface regressions and library churn early. Finally, teams can use difficulty signals from our study to guide staffing: RAG and orchestration benefit from experienced engineers, while onboarding or support teams can handle recurring issues in installation or prompting.

**For Researchers.** Our finding points to several open research opportunity. First, there is a clear need for close the evaluation gap: current benchmark heavily focus on answer quality, while agentic systems demand assessment of schema adherence, plan correctness, grounding, recovery from failure, and full execution traces. Second, research should quantify the impact of churn libraries and SDK updates on breakage and time-to-resolution and propose stability indicators or compatibility manifests for agents. Third, time-to-accept metrics open the door for causal and survival analyses that link tool releases or practices to improved resolution in complex areas

like RAG and orchestration. In multi-agent contexts, correctness remains underexplored; future work should formalize coordination logic, state typing, arbitration, and failure handling, ideally supported by open error taxonomies and execution trace analysis.

**For Educators.** Educators have an opportunity to bring real-world complexity into the classroom. Reproducibility practices (e.g., containers, lockfiles) should be treated as learning outcomes, with students required to resolve actual dependency issues. Lab sequences should reflect the core components of agentic systems: document ingestion, chunking, retrieval QA, schema-aware tool calling, and orchestration graphs with trace inspection. Robustness and observability, not just working demos, should guide evaluation. Students should also learn to evaluate what they build to mirror industry demands, e.g., grounding reports, latency/cost budgets, and regression tests across versions.

## 7 Threats To Validity

**Selection of data source.** Following previous work [4, 5, 10, 18, 31, 37, 63, 75], we use Stack Overflow as our sole data source. This choice is a potential threat, since Stack Overflow may not capture all developer questions and we may miss insights from other venues. However, Stack Overflow is a widely respected community with many participating developers, including experts, which we believe minimizes this risk. In addition, AI Agent is a relatively new area, and it is not easy to find substantive discussions on other platforms.

**Selection of tags.** Relying on tags to collect Stack Overflow questions and answers is a validity threat, as tagging can be incomplete, and some relevant posts may be missed. To reduce this risk, we followed well-established techniques [5, 37, 63, 76]. However, the creation of our tag set could introduce bias, as the selected tags may not cover all AI Agent-related questions, and the final tag set could be influenced by individual experiences. To mitigate this, we engaged the first two researchers in the validation process to enhance the reliability and validity of the final tag selection.

**Construction of topic taxonomy.** We assumed that LDA’s probabilistic topic modeling adequately captures the latent themes in our corpus. While LDA is widely used in software engineering, it has known stochastic variation; to mitigate this, we ran multiple seeds/iterations and selected configurations by coherence to ensure stability [4]. Recently, LLM-based approaches such as BERTOPIC have gained traction [4, 13, 20, 55], leveraging transformer embeddings for richer semantics. However, prior work notes transfer gaps on technical SE text with code/jargon [4, 13, 78] and well-documented challenges on *short* and *small* corpora—exactly our setting (final dataset: 3,191 posts)—including sparse context, clustering/UMAP instability, and variability across domains/languages [26, 32, 52]. Practitioner reports also highlight excessive “outliers” and run-to-run drift in the UMAP–HDBSCAN pipeline [35], and recent analyses link outliers to topic drift in evolving corpora [80]. While refinements (e.g., LLM-guided topic repair) are emerging [22], there is not yet a standard, reproducible recipe for SE datasets of this size. By contrast, LDA has a long record of successful, documented use in SE [15, 16, 63, 65, 76]. Given these trade-offs, we adopt LDA for interpretability, comparability with prior work, and reproducibility, leaving BERTOPIC-style modeling as future work.

**Manual subtopic labeling.** We manually reviewed a stratified, confidence-aware sample (unlike purely random sampling in

much prior work), weighting by each topic's size and the model's per-record confidence, and assigned subtopic labels following established protocols [23, 49, 65, 71]; while some bias is possible, two authors performed analyzing and reconciliation, and a third author independently validated the final labels.

## 8 Conclusion and Future Work

We mapped AI Agent challenges from practitioners' discussions using validated topic modeling method LDA-MALLET, yielding seven topics: (1) *Operations (Runtime & Integration)*, (2) *Document Embeddings & Vector Stores*, (3) *Robustness, Reliability & Evaluation*, (4) *Orchestration*, (5) *Installation & Dependency Conflicts*, (6) *RAG Engineering*, and (7) *Prompt & Output Engineering*. Through manual coding, we further distilled 77 in-depth subtopics. We compare the popularity and difficulty of challenges to prioritize effort and to assess whether community attention matches problem difficulty. We also quantify technology share at scale —LangChain  $\approx$  70.6% with visible roles for LangGraph, Hugging Face, Llama, Azure/OpenAI, and vector stores. Provided a longitudinal view (2021→2025) with event-aligned interpretation, linking mid-2023 take-off to model-API releases. Also, cross-domain comparison shows AI Agents challenges are harder yet competitively popular. Looking ahead, future work should extend this analysis through expert interviews and qualitative methods to provide a complementary perspective and further deepen understanding of emerging developer needs.

## References

- [1] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. 2017. On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology* 88 (2017), 148–158.
- [2] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. 2020. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th international conference on mining software repositories*. 174–185.
- [3] Emad Aghajani, Gabriele Bavota, Mario Linares-Vásquez, and Michele Lanza. 2019. Automated documentation of android apps. *IEEE Transactions on Software Engineering* 47, 1 (2019), 204–220.
- [4] Mumtahina Ahmed, Md Nahidul Islam Opu, Chanchal Roy, Sujana Islam Suh, and Shaiful Chowdhury. 2025. Exploring Challenges in Test Mocking: Developer Questions and Insights from StackOverflow. *arXiv preprint arXiv:2505.08300* (2025).
- [5] Syed Ahmed and Mehdi Bagherzadeh. 2018. What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.
- [6] Nikta Akbarpour, Ahmad Saleem Mirza, Erfan Raoofian, Fatemeh Fard, and Gema Rodríguez-Pérez. 2025. Unveiling Ruby: Insights from Stack Overflow and Developer Survey. *arXiv preprint arXiv:2503.19238* (2025).
- [7] Md Abdullah Al Alamin, Gias Uddin, Sanjay Malakar, Sadia Afroz, Tameem Haider, and Anindya Iqbal. 2023. Developer discussion topics on the adoption and barriers of low code software development platforms. *Empirical software engineering* 28, 1 (2023), 4.
- [8] Reem Alfayez, Yunyan Ding, Robert Winn, Ghada Alfayez, Christopher Harman, and Barry Boehm. 2023. What is asked about technical debt (TD) on Stack Exchange question-and-answer (Q&A) websites? An observational study. *Empirical Software Engineering* 28, 2 (2023), 35.
- [9] Miltiadis Allamanis and Charles Sutton. 2013. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *2013 10th Working conference on mining software repositories (MSR)*. IEEE, 53–56.
- [10] Moayad Alshangiti, Hitesh Sapkota, Pradeep K Murukannaiah, Xumin Liu, and Qi Yu. 2019. Why is developing machine learning applications challenging? a study on stack overflow posts. In *2019 acm/ieee international symposium on empirical software engineering and measurement (esem)*. IEEE, 1–11.
- [11] Ali Asgari, Antonio Guerrero, Roberto Pietrantuono, and Stefano Russo. 2024. From Testing to Evaluation of NLP and LLM Systems: An Analysis of Researchers and Practitioners Perspectives through Systematic Literature Review and Developers' Community Platforms Mining. (2024).
- [12] Ali Asgari, Antonio Guerrero, Roberto Pietrantuono, Stefano Russo, et al. [n.d.]. Adaptive Probabilistic Operational Testing for Large Language Models Evaluation.
- [13] Ibrahim Al Azher, Venkata Devesh Reddy Seethi, Akhil Pandey Akella, and Hamed Alhoori. 2024. Limtopic: Llm-based topic modeling and text summarization for analyzing scientific articles limitations. In *Proceedings of the 24th ACM/IEEE Joint Conference on Digital Libraries*. 1–12.
- [14] Mehdi Bagherzadeh and Raffi Khatchadourian. 2019. Going big: a large-scale study on what big data developers ask. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 432–442.
- [15] Kartik Bajaj, Karthik Patabiraman, and Ali Mesbah. 2014. Mining questions asked by web developers. In *Proceedings of the 11th Working conference on mining software repositories*. 112–121.
- [16] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical software engineering* 19, 3 (2014), 619–654.
- [17] Berkeley Center for Responsible, Decentralized Intelligence. 2025. Agentic AI Summit 2025. <https://rdi.berkeley.edu/events/agentic-ai-summit>. UC Berkeley. Accessed: 2025-10-17.
- [18] Tingting Bi, Peng Liang, Antony Tang, and Xin Xia. 2021. Mining architecture tactics and quality attributes knowledge in stack overflow. *Journal of Systems and Software* 180 (2021), 111005.
- [19] C Bridge. 2011. Unstructured data and the 80 percent rule. *Tersedia di: http://www.clarabridge.com/default.aspx* (2011).
- [20] Wenjuan Bu, Hui Shu, Fei Kang, Qian Hu, and Yuntian Zhao. 2023. Software subclassification based on bertopic-bert-bilstm model. *Electronics* 12, 18 (2023), 3798.
- [21] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657* (2025).
- [22] Shuya Chang, Rui Wang, Peng Ren, Qi Wang, and Haiping Huang. 2024. A Large Language Model Guided Topic Refinement Mechanism for Short Text Modeling. *arXiv preprint arXiv:2403.17706* (2024).
- [23] Zhengpeng Chen, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, Tao Xie, and Xuanze Liu. 2020. A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 750–762.
- [24] Zhiyuan CHEN, Jiakai TANG, Xu CHEN, and Yankai LIN. 2023. A survey on large language model based autonomous agents. *arXiv* (2023).
- [25] William Gemmell Cochran. 1977. *Sampling techniques*. john wiley & sons.
- [26] Muriel de Groot, Mohammad Aliannejadi, and Marcel R Haas. 2022. Experiments on generalizability of BERTopic on multi-domain short text. *arXiv preprint arXiv:2212.08459* (2022).
- [27] Christian Schroeder de Witt. 2025. Open challenges in multi-agent security: Towards secure systems of interacting ai agents. *arXiv preprint arXiv:2505.02077* (2025).
- [28] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. Ai agents under threat: A survey of key security challenges and future pathways. *Comput. Surveys* 57, 7 (2025), 1–36.
- [29] Chenguang Du, Chuqi Wang, Yihan Chao, Xiaohui Xie, and Yong Cui. 2025. AI Agent Communication from Internet Architecture Perspective: Challenges and Opportunities. *arXiv preprint arXiv:2509.02317* (2025).
- [30] Wei Du and Shifei Ding. 2021. A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications. *Artificial Intelligence Review* 54, 5 (2021), 3215–3238.
- [31] Edona Elshan, Olivia Bruhin, Niklas Schmidt, et al. 2024. Unveiling challenges and opportunities in low code development platforms: A StackOverflow analysis. In *Proceedings of the 57th Hawaii International Conference on System Sciences (HICSS)*.
- [32] Yuwei Fan, Lei Shi, and Lu Yuan. 2023. Topic modeling methods for short texts: A survey. *Journal of Intelligent & Fuzzy Systems* 45, 2 (2023), 1971–1990.
- [33] Sally Fincher and Josh Tenenberg. 2005. Making sense of card sorting data. *Expert Systems* 22, 3 (2005), 89–93.
- [34] Jason Gabriel, Geoff Keeling, Arianna Manzini, and James Evans. 2025. We need a new ethics for a world of AI agents. *arXiv preprint arXiv:2509.10289* (2025).
- [35] Maarten Grootendorst and contributors. 2022. BERTopic Issue #461: Discussion on outliers / UMAP-HDBSCAN stability. [https://github.com/MaartenGr/BERTTopic/issues/461](https://github.com/MaartenGr/BERTopic/issues/461). Accessed: 2025-10-13.
- [36] Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. 2020. What do programmers discuss about deep learning frameworks. *Empirical Software Engineering* 25, 4 (2020), 2694–2747.
- [37] Munib Ul Haque, Leonardo Horn Iwaya, and M Ali Babar. 2020. Challenges in docker development: A large-scale study using stack overflow. In *Proceedings of the 14th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*. 1–11.

- [38] Denis Helic and Tiago Santos. 2025. Stack Overflow Is Not Dead Yet: Crowd Answers Still Matter. *arXiv preprint arXiv:2509.05879* (2025).
- [39] Abram Hindle, Anahita Alipour, and Eleni Stroulia. 2016. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering* 21, 2 (2016), 368–410.
- [40] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, Adriane Boyd, et al. 2020. spaCy: Industrial-strength natural language processing in python. (2020).
- [41] Samia Kabir, David N Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2024. Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–17.
- [42] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1-2 (1938), 81–93.
- [43] Noam Kolt. 2025. Governing AI agents. *arXiv preprint arXiv:2501.07913* (2025).
- [44] William H Kruskal. 1957. Historical notes on the Wilcoxon unpaired two-sample test. *J. Amer. Statist. Assoc.* 52, 279 (1957), 356–360.
- [45] Adam P Kubiak and Paweł Kawalec. 2022. Prior information in frequentist research designs: The case of Neyman’s sampling theory. *Journal for General Philosophy of Science* 53, 4 (2022), 381–402.
- [46] Heng Li, Foutse Khomh, Moses Openja, et al. 2021. Understanding quantum software engineering challenges an empirical study on stack exchange forums and github issues. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 343–354.
- [47] Jinrun Liu, Xinyu Tang, Linlin Li, Panpan Chen, and Yepang Liu. 2023. Which is a better programming assistant? A comparative study between chatgpt and stack overflow. *arXiv preprint arXiv:2308.13851* (2023).
- [48] Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. *arXiv preprint cs/0205028* (2002).
- [49] Khalid Mahmood, Ghulam Rasool, Fatima Sabir, and Atifa Athar. 2023. An empirical study of web services topics in web developer discussions on stack overflow. *IEEE Access* 11 (2023), 9627–9655.
- [50] Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. <http://mallet.cs.umass.edu> (2002).
- [51] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.
- [52] Darija Medvecki, Bojana Bašaragin, Adela Ljajić, and Nikola Milošević. 2023. Multilingual transformer and bertopic for short text topic modeling: The case of serbian. In *Conference on Information Technology and its Applications*. Springer, 161–173.
- [53] Tianjun Mo, Ziyi Jiang, and Qichang Zheng. 2025. Interactive AI agent for code refactoring assistance: A study on decision-making strategies and human-agent collaboration effectiveness. *Academia Nexus Journal* 4, 1 (2025).
- [54] Mahshid Naghashzadeh, Amir Haghshenas, Ashkan Sami, and David Lo. 2021. How do users answer matlab questions on q&a sites? a case study on stack overflow and mathworks. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 526–530.
- [55] AmirHossein Naghsanz and Sylvie Ratte. 2023. Enhancing api documentation through bertopic modeling and summarization. *arXiv preprint arXiv:2308.09070* (2023).
- [56] Jerzy Neyman. 1938. Contribution to the theory of sampling human populations. *J. Amer. Statist. Assoc.* 33, 201 (1938), 101–116.
- [57] Jerzy Neyman. 1992. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. In *Breakthroughs in statistics: Methodology and distribution*. Springer, 123–150.
- [58] Moses Openja, Bram Adams, and Foutse Khomh. 2020. Analysis of modern release engineering topics: a large-scale study using stackoverflow-. In *2020 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 104–114.
- [59] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms. In *2013 35th International Conference on Software Engineering (ICSE)*. 522–531. doi:[10.1109/ICSE.2013.6606598](https://doi.org/10.1109/ICSE.2013.6606598)
- [60] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. (2010).
- [61] Zairemmawin Renthlei and C Lallawmkima. [n. d.]. Choosing the Right Sample Size in Social Science Studies: A Methodological Review. *Mizoram Educational Journal* ([n. d.]), 1.
- [62] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*. 399–408.
- [63] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (2016), 1192–1223.
- [64] Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468* (2025).
- [65] Mohammad Mahdi Sayyadnejad, Ali Asgari, Ashkan Sami, and Hooman Tahayori. 2025. Exploring the black box: analysing explainable AI challenges and best practices through stack exchange discussions. *Empirical Software Engineering* 30, 6 (2025), 176.
- [66] Johannes Schneider. 2025. Generative to agentic ai: Survey, conceptualization, and challenges. *arXiv preprint arXiv:2504.18875* (2025).
- [67] Subhasree Sengupta and Caroline Haythornthwaite. 2020. Learning with comments: An analysis of comments and community on Stack Overflow. (2020).
- [68] Manish Shetty, Yinfang Chen, Gagan Somashekhar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shacheer Mishra Gupta, et al. 2024. Building ai agents for autonomous clouds: Challenges and design principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 99–110.
- [69] Amjad Tahir, Aiko Yamashita, Sherlock Licorish, Jens Dietrich, and Steve Counsell. 2018. Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering*. 68–78.
- [70] Fangqiao Tian, An Luo, Jin Du, Xun Xian, Robert Specht, Ganghua Wang, Xuan Bi, Jiawei Zhou, Ashish Kundu, Jayanth Srinivasa, et al. 2025. An outlook on the opportunities and challenges of multi-agent ai systems. *arXiv preprint arXiv:2505.18397* (2025).
- [71] Gias Uddin, Fatima Sabir, Yann-Gaël Guéhéneuc, Omar Alam, and Foutse Khomh. 2021. An empirical study of IoT topics in IoT developer discussions on Stack Overflow. *Empirical Software Engineering* 26, 6 (2021), 121.
- [72] Chao Wang, Zhenpeng Chen, and Minghui Zhou. 2023. Automl from software engineering perspective: Landscapes and challenges. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 39–51.
- [73] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: Survey, landscape, and vision. *Automated Software Engineering* 32, 2 (2025), 1–36.
- [74] Robert F Woolson, Judy A Bean, and Patricio B Rojas. 1986. Sample size for case-control studies using Cochran’s statistic. *Biometrics* (1986), 927–932.
- [75] Wenhua Yang, Chong Zhang, and Minxue Pan. 2023. Understanding the Topics and Challenges of GPU Programming by Classifying and Analyzing Stack Overflow Posts. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1444–1456.
- [76] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. 2016. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology* 31, 5 (2016), 910–924.
- [77] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Chen, and Ahmed E Hassan. 2019. Reading answers on stack overflow: Not enough! *IEEE Transactions on Software Engineering* 47, 11 (2019), 2520–2533.
- [78] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, and Rui Wang. 2023. Unifying the perspectives of nlp and software engineering: A survey on language models for code. *arXiv preprint arXiv:2311.07989* (2023).
- [79] Andy Zou, Maxwell Lin, Eliot Jones, Micha Nowak, Mateusz Dziemian, Nick Winter, Alexander Grattan, Valent Nathanael, Ayla Croft, Xander Davies, et al. 2025. Security challenges in ai agent deployment: Insights from a large scale public competition. *arXiv preprint arXiv:2507.20526* (2025).
- [80] Evangelia Zve, Benjamin Icard, Alice Breton, Lila Sainero, Gauvain Bourgne, and Jean-Gabriel Ganascia. 2025. From Outliers to Topics in Language Models: Anticipating Trends in News Corpora. *arXiv preprint arXiv:2509.22030* (2025).