Universiteit Leiden

AI-Assisted Coding, NDAs, and Confidentiality Boundaries

Ghazaleh Nasrollahi (S3894630)

Supervisors:
Dr. Miros Zohrehvand & Dr. Olga Gadyatskaya

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 2026

Abstract

AI-assisted coding tools such as ChatGPT, GitHub Copilot, Claude, and Cursor are now part of everyday software development, but their use becomes difficult when developers work under non-disclosure agreements (NDAs) and related confidentiality obligations. This thesis studies how developers publicly reason about that difficulty. It analyzes public online discussions from developer communities using a qualitative-priority mixed-method design.

The findings show that developers treat confidentiality as broader than source code alone. They also discuss credentials, repositories, customer data, logs, prompts, and codebase context as potentially sensitive. Disclosure is discussed not only as public release, but also as vendor access, storage, retention, model improvement, and agent access. AI-assisted coding is most often treated as acceptable only when controls are in place, such as local deployment, privacy modes, security review, or limited context sharing.

The thesis concludes that developers frame AI-assisted coding under NDA-like constraints as a problem of controlled exposure: deciding what project information may enter an AI tool, under which safeguards, and with whose responsibility.

Contents

1 Introduction

AI-assisted coding tools have become part of everyday software development. Developers now use systems such as ChatGPT, GitHub Copilot, Claude, and Cursor not only to generate code, but also to explain unfamiliar code, debug errors, review changes, write tests, and reason through implementation problems [Zhang et al., 2023, Hao et al., 2024]. These uses make the tools useful for real programming work. They also mean that developers may need to share details from the project they are working on. A prompt that is more likely to help with a real development task may include source code, logs, repository structure, API details, internal terminology, or information about a client system.

This creates a problem when developers work under NDAs or similar confidentiality rules. An NDA, or non-disclosure agreement, is a legal agreement that limits how confidential information can be shared or used. NDAs existed before generative AI coding tools became common in software development [Hrdy and Seaman, 2024]. For that reason, many NDA clauses do not mention prompts, AI vendors, stored conversations, model training, or coding agents that can access files in a workspace. Instead, they often use broader terms such as "confidential information," "third parties," "disclosure," or "unauthorized use" [Hrdy and Seaman, 2024].

This broad wording can cover more than source code alone, including client data, internal documentation, credentials, business logic, technical designs, and other project information [Hrdy and Seaman, 2024]. However, because the wording is not specific to AI tools, it may not clearly tell a developer whether sending a code snippet, log file, prompt, or repository context to an AI assistant counts as prohibited disclosure. A developer may know that confidential source code should not be published on a public website. The situation is less clear when the same developer wants to paste a short code snippet into ChatGPT, ask Copilot about a private repository, upload an error log to Claude, or let Cursor inspect part of a local project.

The uncertainty is practical because using an AI coding tool can involve sharing information outside the developer's own organization. NDA guidance explains that an NDA sets rules for sharing information in confidence, including what information is covered, how it may be used, and whether it may be shared with others [UK Intellectual Property Office, 2015]. This matters for AI-assisted coding because prompts, uploaded files, repository context, or agent actions may contain project information.

Different AI coding tools have different rules for handling user inputs. These rules can cover storage, review, model improvement, service providers, third-party integrations, retention, and agent actions. For example, OpenAI describes controls for model improvement, temporary chats, retention, and review [OpenAI, 2026a, OpenAI, 2026c]. Anthropic describes inputs and outputs for Claude, including coding and agentic sessions, and explains that data handling can depend on the feature, settings, and third-party services used [Anthropic, 2026]. Cursor's privacy policy describes inputs, suggestions, training choices, service providers, retention, and security limits [Anysphere, 2025]. GitHub presents Copilot through privacy, security, compliance, and transparency controls in its Copilot Trust Center [GitHub, 2026].

The point is that developers cannot answer the NDA question only by asking whether the tool trains on their data. They also have to ask whether project information is sent outside the company,

stored, reviewed, processed by service providers, connected to third-party tools, or accessed by an agent. This is why broad NDA terms such as "disclosure," "third parties," and "permitted use" can be difficult to apply to AI-assisted coding.

These examples do not mean that using an AI coding tool automatically violates an NDA. They show why developers may be uncertain. A developer has to compare broad NDA terms such as "disclosure," "permitted use," and "third parties" with the specific data practices of the AI tool they want to use.

This uncertainty leads directly to the focus of this thesis. If NDA language does not clearly describe AI-assisted coding, then developers have to interpret the boundary themselves. They have to decide which project materials are too sensitive to share, whether using an external AI tool counts as disclosure, and what safeguards would make tool use acceptable.

Existing research helps explain why this issue matters, but it does not fully explain how developers reason through it in practice. Studies of AI coding assistants show that these tools are increasingly used in ordinary development work [Zhang et al., 2023, Hao et al., 2024]. Legal and conceptual work shows that confidentiality obligations can be broad, and that information sharing depends on context, recipients, and transmission conditions [Hrdy and Seaman, 2024, Nissenbaum, 2004]. However, these areas of work are often separate. One explains how developers use AI coding tools, while the other explains why confidentiality and disclosure can be difficult to define. What is still missing is an empirical account of how developers themselves interpret NDA-related boundaries when they discuss AI-assisted coding.

This thesis addresses that gap by analyzing public online discussions among developers. These discussions are useful because they show how developers reason about concrete situations in their own words, including prompts, company code, privacy modes, local deployment, vendor access, and responsibility for acceptable use. The aim is not to decide whether each legal claim made online is correct, but to understand how developers interpret confidentiality boundaries when AI-assisted coding becomes part of practical work.

The study is guided by two research questions:

- **RQ1:** How do software developers interpret what counts as confidential information and disclosure when using AI-assisted coding tools under NDA-like or related confidentiality constraints?
- **RQ2:** How do software developers define acceptable use of AI-assisted coding tools in confidential work settings?

To answer these questions, the thesis uses a qualitative-priority mixed-method design. The main analysis is qualitative because the goal is to understand how developers reason about confidentiality, disclosure, safeguards, and responsibility. Descriptive quantitative counts are used to support that interpretation and to compare patterns across stricter and broader datasets. A reproducible collection pipeline gathered public discussions from Reddit, Hacker News, Stack Exchange, GitHub Issues, DEV.to, and public Discourse communities. After filtering and coding, the main qualitative evidence base was narrowed to a manually reviewed conservative subset, with a broader subset used for comparison. The analysis examines what developers treat as confidential, what they describe

as disclosure risk, what safeguards they consider meaningful, and who they hold responsible for acceptable use.

2 Theoretical Background

This chapter reviews the research needed to understand the thesis topic. It first explains how developers use AI-assisted coding tools in software work. It then explains why public developer discussions are a useful source for studying developer reasoning. After that, it reviews research on developers' privacy and confidentiality challenges more broadly. The chapter then explains why AI-assisted coding can make confidentiality boundaries harder to apply under NDAs or similar rules, before ending with the research gap addressed by this thesis.

2.1 How Developers Use AI-Assisted Coding Tools

AI-assisted coding tools are now used for many parts of software development. Developers use tools such as ChatGPT, GitHub Copilot, Claude, and Cursor to generate code, explain unfamiliar code, debug errors, write tests, review changes, and reason through implementation problems [Zhang et al., 2023, Hao et al., 2024]. This matters for the thesis because these tasks often involve project material, such as source code, logs, documentation, repositories, and client information.

Research on developer interaction with AI coding assistants shows that these tools support different working patterns. Barke et al. identify two patterns, which they call "acceleration" and "exploration" [Barke et al., 2022]. In acceleration, the developer already has a direction and uses the assistant to complete the task more efficiently. In exploration, suggestions help the developer examine alternatives while deciding how to proceed. This distinction matters because exploratory work can require more project context. A general syntax question may reveal little, whereas a real bug, failing test, or architecture problem may require details about the actual system.

Workplace research also shows that repeated use of generative AI tools can change how developers experience their work. In a workplace experiment and diary study, Butler et al. observed that regular access led participants to rate the tools as more useful and enjoyable, without a corresponding increase in confidence in generated code [Butler et al., 2024]. This matters here because usefulness and concern can exist at the same time. Developers may find AI coding tools helpful in everyday work while still being unsure when it is appropriate to use them with confidential project information.

2.2 Developer Discourse as a Source for Software Engineering Research

Software-engineering research often uses public developer discussions to study how developers understand practical problems. These discussions are useful because they do not only show final answers. They also show how developers explain a problem, what details they treat as important, how others respond, where they disagree, and what solutions they suggest. This makes public developer discussions useful for studying developer reasoning, not only developer behavior.

Several recent studies show how this kind of data can be used. Zhu et al. analyze comments

connected to Stack Overflow questions and show that these exchanges carry information about the question-answering process that would be lost if only questions and answers were retained [Zhu et al., 2022]. El Aoun et al. use Stack Exchange forums and GitHub Issues to study quantum software engineering challenges, showing that public technical discussions can reveal both the problems developers face and the kinds of support they need [El Aoun et al., 2022]. Alamin et al. use developer discussions to study low-code software development challenges, while Croft et al. use developer discussions to examine security challenges across programming languages [Al Alamin et al., 2021, Croft et al., 2021]. Together, these studies support using public developer discourse to examine practical problems and recurring reasoning across developer communities.

This thesis uses similar public sources, but for a more specific topic. It studies discussions about AI-assisted coding under NDA-like confidentiality constraints. The focus is on how developers discuss what project information may be confidential, what kinds of AI-tool use may count as disclosure, what safeguards they consider useful, and when they treat AI-assisted coding as acceptable.

This matters because formal NDA wording and official tool policies do not show how developers understand these rules in everyday work. Public developer discussions can show that reasoning more directly. For example, a forum thread may show developers debating whether a prompt counts as sharing information, whether code sent to an AI vendor is different from code posted publicly, whether privacy mode is enough, or whether local deployment changes the risk.

The gap is that existing developer-discourse research has not fully examined this specific combination: AI-assisted coding tools, NDA-like confidentiality constraints, and developers' own reasoning about disclosure and acceptable use. This thesis contributes to that area by using public developer discussions to study how developers interpret those boundaries in practice.

2.3 Developers, Privacy, and Confidentiality Challenges

Research on developers and privacy shows that developers often have to turn broad privacy or security expectations into concrete software decisions. This is not always easy. Rather than placing privacy work entirely on individual programmers, Tahaei et al. distribute the needed support across tool designers, organizations, educators, and regulators [Tahaei et al., 2022]. This matters for the present thesis because developers cannot manage information-protection duties through individual judgement alone. They also depend on suitable tools, policies, documentation, and organizational support.

Other studies show similar difficulties in specific development settings. Tahaei et al. found that developers and users can understand app permissions differently, and that developers may request permissions because of unclear requirements or third-party library needs [Tahaei et al., 2023]. Delile et al. studied privacy-related questions on Stack Overflow and show that developers use public forums to ask about privacy problems such as consent, identification, and data aggregation [Delile et al., 2023]. Horstmann et al. found that professional developers struggled to implement privacy-compliant solutions even when working on privacy-sensitive tasks [Horstmann et al., 2025].

These studies are not about NDAs directly, but they are relevant because they show a broader pattern: developers often face difficulty when legal, privacy, or security expectations have to be applied in technical work. Confidentiality under NDAs creates a similar problem. A developer may

understand that some information must be protected, but still be unsure how that duty applies to a specific tool, workflow, or data flow. In AI-assisted coding, this uncertainty becomes easier to see because project information may be placed into prompts, sent to vendors, stored in chat histories, used for model improvement, or accessed by coding agents.

These studies help frame this thesis, but they do not answer the two research questions asked here. This thesis does not study privacy engineering in general. It studies how developers discuss two specific issues: what counts as confidential information or disclosure when AI-assisted coding tools are used under NDA-like constraints, and what conditions make that use acceptable in confidential work settings.

2.4 Why AI Tool Use Can Affect Confidentiality Boundaries

AI-assisted coding often depends on project context. A useful answer may require more than a general programming question. Prior work shows that developers use AI coding tools not only for code generation, but also for debugging, explanation, review, and problem solving [Barke et al., 2022, Hao et al., 2024]. These tasks may require project-specific information. A developer may provide a code snippet, a stack trace, a database schema, an API response, a repository structure, documentation, or a description of internal business logic. Research on repository-level prompting also shows that code models can benefit from context taken from other files and from the wider repository structure [Shrivastava et al., 2022]. In ordinary development, these materials may stay inside the team or organization. When they are entered into an AI tool, they may become part of a prompt, a chat history, an external service request, or an agent workflow.

This matters for confidentiality because an information flow is shaped not only by its content, but also by where the information goes, who can access it, and under what conditions it is transmitted. Contextual integrity evaluates an information flow by considering its social context, the actors involved, the type of information, and the conditions governing transmission [Nissenbaum, 2004]. In this thesis, this means that confidentiality concerns are not limited to sharing a complete private repository. Smaller pieces of project context may also reveal protected information. For example, a stack trace may expose internal service names, a database schema may reveal product design, a short code fragment may contain proprietary logic, and a prompt may combine several details about a confidential system. This interpretation is also consistent with legal scholarship showing that confidentiality obligations may protect more than trade-secret source code alone [Hrdy and Seaman, 2024].

This connection is central to the thesis. If an AI tool becomes part of everyday work, developers may start to treat some forms of sharing as normal, especially when the tool is useful, common, or approved by others. At the same time, they may draw boundaries around materials they consider too sensitive, such as credentials, customer data, private repositories, or logs. The thesis studies how developers draw this boundary: what project information they treat as confidential, what forms of AI-tool use they understand as disclosure, and what conditions or safeguards they use to define acceptable use.

2.5 NDAs and Broad Confidentiality Duties

An NDA, or non-disclosure agreement, sets conditions for how specified confidential information may be shared and used. NDA guidance explains that these agreements normally identify what information is protected, who may receive it, and for what purpose it may be used [UK Intellectual Property Office, 2015]. Legal scholarship also shows that confidentiality agreements may extend beyond trade-secret law and restrict both the disclosure and use of protected information [Hrdy and Seaman, 2024]. Therefore, protected information may include not only source code, but also client data, credentials, internal documentation, business logic, technical designs, and other project information.

The difficulty is that these broad confidentiality duties do not always describe AI-assisted coding workflows directly. They may not expressly address prompts, external AI providers, retained conversations, model improvement, tool telemetry, or coding agents that can inspect workspace files. GitHub Copilot became generally available in June 2022, and OpenAI introduced ChatGPT in November 2022 [GitHub, 2022, OpenAI, 2022]. This does not mean that existing NDAs are outdated or that every use of an AI tool is forbidden. It means that broad confidentiality terms must now be interpreted in relation to newer technical data flows.

As a result, a developer may understand the general duty not to disclose confidential information but still be uncertain about how that duty applies to an AI tool. For example, an NDA may prohibit disclosure to third parties without explaining whether sending a prompt to an external AI provider counts as disclosure. It may protect source code, client data, and internal documentation without specifically mentioning stack traces, repository context, or agent access to local files. The central problem is therefore that NDA language is broad, while AI-assisted coding involves specific and tool-dependent ways of processing project information.

3 Research Design and Methodology

3.1 Research Design

This thesis uses a qualitative-priority mixed-method design. The main purpose is to study how developers publicly reason about AI-assisted coding under NDAs or similar confidentiality rules. The research questions focus on NDA-related boundaries: what project material developers treat as confidential, what kinds of AI-tool use they describe as disclosure or exposure, what safeguards they mention, and what conditions make AI-assisted coding acceptable or unacceptable in confidential work settings. These questions require attention to meaning and context, so the main analysis is qualitative. The methodology was designed for this thesis. It was not copied from one existing algorithm or one previous study. Instead, it combines established research methods with a custom data pipeline built for this topic. The established parts are qualitative content analysis, text-as-data filtering, LLM-assisted deductive coding, manual review, and descriptive comparison. The custom parts are the query bank, the relevance filters, the coding schema, and the dataset layers, because these had to match the specific research problem: AI-assisted coding, NDAs, confidentiality, disclosure, safeguards, and acceptable use.

The study also reports descriptive counts. These counts show how often selected patterns appeared in the collected discussions, such as vendor access, credentials, privacy modes, approval, or conditional acceptance. They are useful because they make the qualitative interpretation more transparent: the reader can see which patterns appeared often and which appeared less often. Because the corpus was built through targeted searches, the counts describe the collected material rather than the wider developer population. This follows text-as-data guidance that computational analysis should fit the research design and that researchers should be clear about what their data can and cannot show [Grimmer et al., 2022]. The final interpretation remains qualitative, with counts used to support and compare the observed patterns.

The four dimensions show how broad NDA-like terms were turned into questions that could be examined in developer discussions. An NDA usually protects "confidential information," meaning information that the company, client, or project owner does not want shared outside an allowed setting. It may also restrict "disclosure," meaning making that information available to someone who is not allowed to receive it. A "third party" usually means an outside person or organization, which may include a tool provider if project information is sent to an external AI service. "Permitted use" means using the information only for an allowed purpose, such as completing the project work. "Unauthorized use" means using or sharing the information outside those allowed purposes. These terms are important for the thesis, but they are too broad to code directly. For that reason, the thesis translated them into four practical dimensions [UK Intellectual Property Office, 2015, Hrdy and Seaman, 2024].

`D1` covers confidential information and boundary definition. It translates the NDA question of what counts as "confidential information" into a practical coding question: what project material do developers treat as sensitive? This includes source code, repositories, logs, credentials, client data, documentation, architecture details, prompts, and codebase context. This dimension is also supported by generative-AI risk guidance, which treats data privacy, sensitive data, information security, and intellectual property as important risk areas [National Institute of Standards and Technology, 2024].

`D2` covers disclosure, vendor access, retention, policy, and compliance reasoning. It translates the NDA question of what counts as "disclosure" or sharing with a "third party" into a practical coding question: what happens when project information is entered into an AI tool? This includes where the information goes, who may access it, whether it may be stored or reviewed, whether it may be used for model improvement, and whether company or client policy allows that use. This dimension is supported by NIST guidance, which connects generative-AI risk to the data involved, third-party services, and organizational controls [National Institute of Standards and Technology, 2024]. It is also supported by provider data-control policies, which show that user inputs can be governed by tool-specific rules about storage, review, model-improvement settings, and retention [OpenAI, 2026a].

`D3` covers mitigation practices. It translates the NDA concern with avoiding unauthorized disclosure into a practical coding question: what do developers suggest doing to reduce confidentiality risk? This includes redacting details, using pseudocode, minimizing snippets, removing credentials, using local or self-hosted models, enabling enterprise privacy settings, excluding files, or asking security teams for approval. This dimension is supported by LLM security guidance, which identifies sensitive information disclosure, prompt injection, improper output handling, and excessive agency as important risks in LLM applications [OWASP Foundation, 2025]. It is also supported by NIST guidance, which connects generative-AI risk management to software development, IT governance, legal, com-

pliance, and risk-management processes [National Institute of Standards and Technology, 2024].

D4 covers norms and expectations about acceptable use. It translates the NDA question of "permitted use" into a practical coding question: when do developers describe AI-assisted coding as acceptable, unacceptable, unclear, risky, or allowed only under specific conditions? This matters because the thesis is not only about formal NDA wording. It is also about how developers discuss acceptable use in everyday work. Recent workplace research shows that regular use of generative-AI coding tools can change developers' views of usefulness and work practice, while trust concerns can remain [Butler et al., 2024]. Other software-engineering research also shows that generative AI can change development workflows and collaboration patterns [Ulfsnes et al., 2024].

Together, the four dimensions connect the research questions to the implemented method. D1 and D2 support RQ1, because they examine what developers treat as confidential information and how they describe disclosure risk. D3 and D4 support RQ2, because they examine the safeguards, norms, and conditions through which developers define acceptable use. These dimensions shaped the query bank, the lexical relevance filters, and the coding schema, so that collection, filtering, and analysis followed the same logic.

In simple terms, the method follows this chain:

Stage	**What happened**
Collection	Public discussions were collected from six sources
First filtering	Clearly irrelevant material was removed
Second filtering	Two filtered corpora were created
Coding	Each response was coded with the fixed schema
Review	Weak or irrelevant coded rows were removed
Analysis	Codes were counted and interpreted

3.2 Data Source and Rationale

This thesis uses public online developer discussions as its data source. This choice fits the research design because the study focuses on how developers reason about NDA-related boundaries in practice, not on what an NDA would legally mean in a specific court case. Public discussions make this reasoning visible because they contain questions, replies, disagreements, warnings, examples, and suggested safeguards.

The data comes from six public sources: Reddit[1], Hacker News[2], Stack Exchange[3], GitHub Issues[4], DEV.to[5], and public Discourse communities[6]. These sources were chosen because they capture different kinds of developer discussion. Reddit and Hacker News often include informal debate

[1] https://www.reddit.com/
[2] https://news.ycombinator.com/
[3] https://stackexchange.com/
[4] https://github.com/features/issues
[5] https://dev.to/
[6] https://www.discourse.org/

about workplace rules, privacy, security, and tool use. Stack Exchange contains question-and-answer discussions around practical programming and workplace problems. GitHub Issues contain repository-centered discussions about tools, features, bugs, and project workflows. DEV.to contains longer developer posts and comment threads. The selected public Discourse communities included tool-focused forums, especially Cursor Forum and OpenAI Community, where users discuss AI coding tools and related workflows.

Using these sources is consistent with software-engineering research that studies public technical discussions to understand developer reasoning and practical challenges [Zhu et al., 2022, El Aoun et al., 2022]. In this thesis, similar public sources are used for a more specific purpose: to study discussions about AI-assisted coding, NDAs, confidentiality, disclosure, vendor access, privacy modes, local deployment, and acceptable use.

This use of public discussion data is informed by netnographic research. Netnography studies online communities by reading their discussions in context, rather than treating each comment as an isolated sentence [Kozinets, 2019]. This matters here because many developer comments are short. A reply such as "this would violate company policy" or "use a local model instead" only makes sense when it is read together with the thread title, the original question, and the AI tool being discussed. For that reason, the unit of analysis is the individual response or comment, but each response is stored together with its parent question or thread opener.

The collection window starts on January 1, 2022. This date was chosen because generative AI coding tools became much more visible in everyday software development from 2022 onward. GitHub Copilot became generally available in June 2022, and ChatGPT was introduced in November 2022 [GitHub, 2022, OpenAI, 2022]. Starting the collection in 2022 therefore keeps the dataset focused on the period in which developers were beginning to discuss these tools as part of ordinary coding work. The collection ended on March 31, 2026.

The dataset was built from public material only. However, public availability does not remove ethical responsibility. Internet-research ethics guidance emphasizes that public online data should still be handled with care, especially when the topic involves sensitive work situations or possible legal obligations [Franzke et al., 2020]. For this reason, the thesis does not focus on identifying individual users. The analysis looks for repeated patterns across discussions rather than personal judgments about specific developers.

3.3 Data Collection and Corpus Construction

After the data sources were selected, the dataset was built through a reproducible Python collection pipeline. The aim of the pipeline was to collect public discussions that connected AI-assisted coding with NDAs, confidentiality, disclosure, safeguards, or acceptable use. The collection did not search for AI coding discussions in general. It searched for discussions that were likely to contain reasoning relevant to the two research questions.

The collection started from a query bank. The query bank contained 15 query groups, with five queries in each group, for a total of 75 search queries. These queries were defined before data collection and were built from the two research questions and the four operational dimensions. Because no existing query bank matched this topic, the queries were built from the concepts used

in the thesis: AI-assisted coding tools, confidential project material, NDA-related boundaries, disclosure risk, safeguards, and acceptable use. This follows the logic of directed qualitative content analysis, where the study begins with concepts from the research questions and then collects material that can answer them [Hsieh and Shannon, 2005]. It is also consistent with software-engineering studies that use public developer discussions to study practical reasoning and developer challenges [Zhu et al., 2022, El Aoun et al., 2022, Al Alamin et al., 2021, Croft et al., 2021]. The complete query bank, including all 15 groups and all 75 queries, is reproduced in Appendix C.

Each query combined terms from three areas: AI-assisted coding tools, project or confidential material, and a boundary issue. The AI-tool terms included examples such as *ChatGPT*, *Copilot*, *Claude*, *Cursor*, and other coding assistants. The project-material terms included examples such as *company code*, *client code*, *private repository*, *stack trace*, *API keys*, and *database schema*. The boundary terms included examples such as *NDA*, *confidential*, *policy*, *retention*, *training*, *privacy mode*, *sanitize*, and *acceptable use*. This design made the query bank broad enough to capture different ways developers might discuss the issue, while still keeping the collection focused on the thesis topic.

The query groups were aligned with the four operational dimensions. Some queries focused on confidential material, such as company code, client code, private repositories, logs, database schemas, and API keys. These queries supported `D1`. Other queries focused on disclosure and policy questions, such as NDA violations, company policy, vendor retention, model training, and whether AI providers store prompts. These queries supported `D2`. A third set focused on safeguards, such as sanitizing code, redacting identifiers, using pseudocode, limiting snippets, or using local models. These queries supported `D3`. A final set focused on norms and acceptable use, such as whether developers treat AI coding tools as normal at work or only acceptable under certain rules. These queries supported `D4`. Illustrative examples are shown in Table 1. The full query bank was kept in the project repository as part of the reproducible collection pipeline.

Dimension	**Illustrative query examples**
`D1` Confidential material	`can i paste company code into chatgpt`; `stack trace chatgpt confidential`; `customer data in ai coding assistant`
`D2` Disclosure and vendor risk	`does using copilot violate nda`; `does cursor store company code`; `does copilot retain private repository code`
`D3` Safeguards	`sanitize code snippet before chatgpt`; `pseudocode instead of source code llm`; `mask identifiers before using cursor ai`
`D4` Acceptable use and norms	`acceptable use of ai coding tools with internal code`; `team rules for github copilot confidential code`; `is it okay to use claude code at work`

Table 1: Illustrative examples from the query bank.

For each enabled source, the pipeline selected the relevant queries from the query bank and collected public threads or posts through source-specific access methods. Reddit was collected through public search and thread JSON endpoints, with PullPush-supported discovery where needed. Hacker News was collected through Algolia search and the official item API. Stack Exchange was collected through advanced search and answer endpoints. GitHub Issues were collected through issue search and

issue comments. DEV.to was collected through article and comment endpoints. Public Discourse communities were collected through search and topic JSON endpoints.

The unit of analysis is the individual response or comment. This means that one question or thread could contribute more than one row to the dataset if it had several relevant answers or comments. Each answer or comment was stored as a separate response record, but it was linked back to the parent question or thread opener. This allowed the analysis to examine individual responses while still keeping the context needed to understand them.

For each response record, the dataset keeps the response text, the parent question or thread opener, the thread title, the source, the forum or community, the query that found the thread, and the collection method. This was important because many comments are short and only become meaningful when read with the question or thread that prompted them.

The pipeline also stored provenance information for each response record. Each retained response includes a run identifier, collection timestamp, source name, thread URL, query identifier, query group, query text, response identifier, response type, and basic metadata such as creation time and score when available. Every retained response was therefore traceable back to its source, query, and collection run.

Before relevance filtering, the pipeline cleaned and organized the collected rows. It removed duplicate responses within each source and query, normalized repeated text, and removed very short responses below the minimum response length of 60 characters. These steps reduced obvious noise before the more substantive relevance filters were applied. The next section explains those filtering rules and the final corpus selection.

3.4 Relevance Filtering and Dataset Selection

After collection, deduplication, the 60-character minimum, and the first-stage lexical relevance filter, the pipeline retained 1,457 responses from 283 threads across six public sources. This was the broad first-filter corpus, not every raw item returned by the platform search interfaces. The high-signal and strict corpora were then created as parallel second-stage selections from this corpus.

The filtering had two stages. The first stage checked whether a response was generally relevant to the thesis topic. The second stage then created two filtered datasets from the material that passed the first stage: a broader *high-signal corpus* and a stricter *strict corpus*. The high-signal corpus was designed to keep more potentially relevant material. The strict corpus was designed to keep only responses with clearer response-level evidence.

The first filtering stage used a rule-based lexical relevance scorer. This means that the pipeline compared the combined parent question and response text against predefined lists of terms. The first stage worked in three steps. First, the pipeline searched the text for different groups of terms, such as AI-tool terms, confidentiality terms, software-work terms, legal terms, mitigation terms, and acceptable-use terms. Second, it checked whether the required groups appeared together. Third, it assigned a relevance score and relevance tier based on the strength of the match.

A response was kept by the first filter only if the combined parent question and response text contained four required types of evidence. First, the text had to mention an AI tool or AI-assisted

coding, such as ChatGPT, Copilot, Claude, Cursor, LLMs, or AI assistants. Second, it had to mention confidentiality, such as confidential, private, proprietary, NDA, client data, customer data, secret, or sensitive. Third, it had to include software-work context, such as code, repository, stack trace, API, database, schema, logs, project, or work. Fourth, it had to include a boundary issue, such as disclosure, legal risk, company policy, approval, retention, model training, vendor access, sanitization, privacy mode, or acceptable use. In other words, the first filter kept material that connected AI-assisted coding with confidentiality and with a practical boundary question.

The filter also recorded why each response was kept. For each response, the pipeline stored the matched AI terms, confidentiality terms, code or work terms, boundary terms, legal terms, mitigation terms, normative terms, and engineer-discourse terms. This made it possible to see which signals caused the response to pass the filter. Appendix A lists every keyword category, its scoring weight, and the auxiliary and exclusion terms used by the first-stage relevance filter.

The relevance score was calculated by adding points for different kinds of evidence found in the combined question-response text. AI-tool language and confidentiality language each added three points. Code language and work-context language each added two points. Boundary, legal, and mitigation language each added two points. Normative language, engineer-discourse language, longer responses, and comment-type responses each added one point. A response was marked as relevant only if it contained AI-tool language, confidentiality language, code or work context, and at least one boundary, legal, mitigation, normative, or policy signal.

The relevance tier was assigned from this score. If a response met the inclusion rule and scored 12 or higher, it was marked as `high`. If it met the inclusion rule but scored below 12, it was marked as `medium`. If it did not meet the inclusion rule but still scored 8 or higher, it was marked as `review`. All other responses were marked as `discard`. Only responses marked as `high` or `medium` were kept for the second filtering stage.

The first filter also produced matched dimension labels. These labels were based on the kinds of terms found in the combined text. If the text discussed what project material might be confidential, it was marked as related to `D1`. If it discussed policy, legal risk, vendor access, retention, or model training, it was marked as related to `D2`. If it discussed safeguards such as sanitization, redaction, local models, privacy settings, or approval, it was marked as related to `D3`. If it discussed whether AI use was normal, acceptable, risky, or allowed under conditions, it was marked as related to `D4`. These labels were not final findings. They were used to track why a response was relevant and to connect filtering with the later coding schema.

A rule-based filter was used because the thesis needed clear inclusion rules. Each kept response had to show a visible connection between AI-assisted coding and confidentiality. This made the dataset easier to check and kept the material connected to the research questions. It also avoided using a more complex classifier whose decisions would be harder to explain. This follows text-as-data guidance that computational filtering should fit the research design and that the limits of the resulting dataset should be made clear [Grimmer et al., 2022].

The second filtering stage was used to create two versions of the relevant corpus. Both versions started from responses that had passed the first filter, but they used different strictness levels.

The broader version produced the *high-signal corpus*. Its algorithm was more inclusive. It checked whether the title or parent question contained enough AI and confidentiality context, and it applied

source-specific rules to remove obvious noise. For example, Discourse results were kept when the title or parent question contained terms such as privacy, repository, security, policy, retention, privacy mode, local mode, file exclusion, context, memory, enterprise, redaction, or agent-related terms. GitHub Issues and Reddit used their own inclusion terms, such as policy, sanitized, code, security, private, privacy, or work. The logic of this version was to keep a discussion if the surrounding thread showed clear relevance, even if some individual responses were less detailed. This produced the high-signal corpus of 671 responses from 170 threads.

The stricter version produced the *strict corpus*. Its algorithm was more demanding. A response entered the strict corpus only if it passed three checks. First, the response itself had to contain relevant terms, such as confidential, private, company, client, repository, source code, prompt, retention, training, policy, security, approval, sanitization, privacy mode, or local mode. Second, the thread title had to mention an AI tool or AI-assisted coding. Third, the title or parent question had to connect at least two of the main issue areas: confidential material, disclosure or boundary concerns, and risk or policy concerns. This means that the strict version did not keep a response only because it appeared in a relevant thread. The response itself also had to contain evidence connected to the thesis topic. This produced the strict corpus of 237 responses from 78 threads. Appendix B lists the complete title, parent-question, response, and source-specific keyword sets used for the high-signal and strict selections.

The difference between the two versions is therefore the inclusion rule. The high-signal corpus kept discussions that were likely to be relevant at the thread level. The strict corpus kept responses that were clearly relevant at the individual-response level. For that reason, the high-signal corpus was used as a broader comparison dataset, while the strict corpus was used as the starting point for the main qualitative coding.

Both final corpora retained five sources after filtering. Six sources were searched at the collection stage, but DEV.to did not remain in the final corpora because its candidate material did not pass the final filtering rules. After filtering, the dataset contained only discussions that clearly connected AI-assisted coding with confidentiality, NDAs, disclosure risk, safeguards, or acceptable use. This means the analysis is based on material that directly matches the research questions.

Table 2 reports the source composition at each filtering layer. Each cell gives the number of responses followed by the number of distinct threads in parentheses. The high-signal and strict columns are parallel selections from the same 1,457-response first-filter corpus; the strict corpus was not created by filtering the high-signal corpus.

Source	First filter responses (threads)	High-signal responses (threads)	Strict responses (threads)
DEV.to	1 (1)	0 (0)	0 (0)
Discourse	694 (185)	262 (114)	101 (51)
GitHub Issues	263 (23)	9 (5)	8 (4)
Hacker News	420 (46)	375 (41)	125 (20)
Reddit	61 (11)	22 (7)	1 (1)
Stack Exchange	18 (17)	3 (3)	2 (2)
Total	**1,457 (283)**	**671 (170)**	**237 (78)**

Table 2: Source-by-source corpus size across the broad first filter and the two parallel second-stage selections.

The difference between the two versions is therefore the inclusion rule. The high-signal corpus kept discussions that were likely to be relevant at the thread level. The strict corpus kept responses that were clearly relevant at the individual-response level. For that reason, the high-signal corpus was used as a broader comparison dataset, while the strict corpus was used as the starting point for the main qualitative coding.

Both final corpora retained five sources after filtering. Six sources were searched at the collection stage, but the one DEV.to response in the first-filter corpus did not remain in either parallel second-stage corpus. The source counts also show that the resulting corpora were not balanced samples of the six platforms: Discourse and Hacker News supplied most retained responses. This source composition is considered again in the limitations section.

3.5 Qualitative Coding and Manual Review

After the relevant corpora were created, the next step was qualitative coding. The purpose of coding was to connect each response in the dataset to the research questions. Each response was treated as one row of evidence. A single thread could contain several responses, and each response was coded separately. This was important because different people in the same thread could discuss different issues, such as source code, vendor access, privacy settings, company approval, or NDA risk.

The coding schema was built around the two research questions and the four operational dimensions. `D1` was used for confidential information and boundary definition. `D2` was used for disclosure, vendor access, retention, policy, and compliance. `D3` was used for safeguards and mitigation. `D4` was used for norms and acceptable use. In simple terms, `D1` and `D2` help answer RQ1, while `D3` and `D4` help answer RQ2.

The schema also included more detailed labels. For confidential material, a response could be coded as discussing source code, repositories, client data, credentials, logs, documentation, architecture, prompts, or memory. For risk, a response could be coded as discussing vendor access, retention or training, internal policy, customer privacy, credential exposure, code leakage, or NDA-related

concern. For safeguards, a response could be coded as discussing redaction, pseudocode, snippet minimization, local models, privacy modes, approval, security review, or avoiding use. The schema also captured whether the response treated AI-assisted coding as acceptable, unacceptable, allowed only with controls, dependent on company policy, or unclear.

These coded fields are the basis for the findings section. For example, if a response discussed API keys or secrets, it could receive the code `credentials_or_secrets`. If it discussed an AI provider seeing, storing, or processing project information, it could receive a risk code such as `vendor_access` or `retention_or_training`. If it suggested using a local model, it could receive the mitigation code `local_or_self_hosted_model`. If it said AI-assisted coding was acceptable only with approval or privacy settings, it could receive the norm code `allow_only_with_controls`. The results tables are calculated from these row-level codes. A count shows how many responses received a code, and the percentage is calculated by dividing that count by the number of responses in the subset.

Some code fields allowed more than one label for the same response. For example, one response could discuss both vendor access and retention, or both approval and local deployment. For this reason, percentages for risk, mitigation, and responsibility categories do not need to add up to 100%. The counts are used to show which patterns appeared often in the collected discussions.

The first coding pass was supported by an LLM. The completed coding runs used `openai/gpt-5.4-20260305` via OpenRouter. For each response, the model received the thread title, the parent question or thread opener, the response text, and the relevance labels from the filtering stage. It then returned only structured labels from the fixed coding schema, together with short evidence spans and a short memo. The model was not asked to write the findings or decide the thesis conclusion. Its role was to apply the same set of labels to many responses in a consistent format.

This coding approach was chosen because the task needed more than keyword matching. A keyword-only method could show that a response mentioned words such as "ChatGPT," "NDA," or "company code," but it could not show how the response used those words. The same words could appear in a warning, a policy question, a mitigation suggestion, or a comment saying that AI tool use is acceptable under certain conditions. A supervised machine-learning classifier was also not suitable because the project did not begin with a large manually labelled training set. Recent work on LLM-assisted qualitative coding shows that LLMs can support deductive coding when a codebook is defined in advance, but that the outputs still require researcher review and interpretation [Chew et al., 2023, Xiao et al., 2023, Oksanen et al., 2025].

The project therefore used schema-constrained LLM-assisted coding rather than free-form LLM summaries. The schema forced the model to choose from predefined fields and allowed values. This made the output easier to compare across responses and easier to check during review. Structured-output documentation also supports this design because a JSON schema can require the model output to follow the same fields and allowed values each time [OpenAI, 2026b].

The strict corpus was coded first because it had stronger relevance requirements. It contained 237 responses from 78 threads. After the first coding pass, 120 responses were labelled as `Neither`, meaning that they did not clearly answer either research question. These rows were removed. This left 117 substantive responses from 44 threads. These were the responses that clearly discussed confidential information, disclosure risk, safeguards, acceptable use, or responsibility in relation to

AI-assisted coding.

A manual review was then used to check the quality of this 117-response subset. The review had two parts. First, 14 responses were selected from different sources, research-question labels, and confidence levels. These 14 responses were checked to see whether the assigned codes matched the actual text. Second, all four responses marked as low confidence were checked separately. These four responses were not a sample; they were the complete low-confidence group. They were removed because they were too weak for the final evidence base. Their surrounding threads were relevant, but the responses themselves did not clearly discuss the research questions. After this step, the final conservative strict subset contained 113 responses from 43 threads. This subset is the main evidence base used in the findings section.

The broader comparison subset followed the same logic. The high-signal corpus started with 671 responses from 170 threads. After coding and removing responses labelled as `Neither`, the high-signal substantive subset contained 240 responses from 72 threads. After the conservative review step, the high-signal conservative subset contained 229 responses from 71 threads. This broader subset was not used as the main evidence base. It was used only for comparison. If the same pattern appeared in both the strict subset and the broader subset, the finding was less likely to depend only on one narrow filtering choice.

The final findings therefore come from a clear chain: public responses were collected, filtered for relevance, coded with a fixed schema, reviewed, and then counted and interpreted. The strict conservative subset is used for the main interpretation because it is the cleanest and most reviewed dataset. The high-signal conservative subset is used to check whether the same patterns remain visible in a broader dataset. The coding does not decide whether a developer's legal claim about an NDA is correct. It shows how developers in the collected discussions reasoned about confidential information, disclosure, safeguards, responsibility, and acceptable use.

3.6 Descriptive Counts and Comparison Strategy

The thesis uses descriptive counts to support the qualitative analysis. These counts show how often each code appeared in the coded material. For example, they show how often developers discussed vendor access, internal policy, credentials, local models, privacy modes, approval processes, or conditional acceptance. The counts help show which patterns appeared often in the corpus and which appeared less often.

The counts are not used as survey results. They do not show what percentage of all developers think a certain way. The dataset was collected through targeted searches and relevance filtering, not through random sampling. For that reason, the counts describe only the collected corpus. They help answer questions such as which risks were emphasized, which safeguards were mentioned, and whether the same pattern appeared in both the stricter and broader datasets.

The main analysis uses the final conservative strict subset because it has the clearest response-level evidence. The broader high-signal conservative subset is used only as a comparison layer. If a pattern appears in both subsets, the finding is less likely to depend only on one narrow filtering choice.

The comparison focuses on the codes that are most directly connected to the research questions.

For RQ1, the analysis compares confidential artifacts, boundary positions, disclosure risks, and the presence of `D1` and `D2`. For RQ2, it compares mitigation types, norm stances, responsibility attribution, and the presence of `D3` and `D4`. This keeps the quantitative description connected to the same analytical structure used in collection and coding.

The findings section therefore reports both qualitative interpretation and descriptive counts. The interpretation explains how developers reason about confidentiality and acceptable use. The counts show how often the coded patterns appear in the collected material. Together, they support cautious claims about recurring patterns in public developer discourse, without claiming to measure the views of all developers.

3.7 Methodological Limitations and Ethics

The method has several limitations. First, the data comes from public online discussions. This means the thesis studies visible public reasoning, not private workplace practice. Developers may discuss a concern online in a simplified way, or they may leave out details about their employer, client, contract, or internal policy. The findings therefore show how confidentiality is discussed in public developer spaces, not exactly how every developer behaves at work.

Second, the corpus is query-driven. This was useful because the searches stayed close to the research questions, but it also means the dataset is shaped by the chosen keywords. Discussions that used different wording may have been missed. To reduce this problem, the query bank included different types of terms, including tool names, NDA terms, confidentiality terms, source-code terms, vendor-retention terms, and mitigation terms. Even so, the dataset should be understood as a focused corpus rather than a complete collection of every relevant discussion.

Third, the final evidence base is not evenly distributed across sources. In the conservative strict subset, 76 responses come from Hacker News, 29 from Discourse, and 8 from GitHub Issues. This matters because different platforms have different discussion cultures. Hacker News often contains security- and policy-oriented debate, while Discourse forums often contain tool-specific user discussions. The findings should therefore be read as patterns in the collected public developer discourse, not as a balanced picture of all developer communities.

Fourth, the coding was partly supported by an LLM. The LLM was used to help organize the data by applying the same coding categories to each response. However, the LLM was not treated as the final analyst. The coded results were reviewed, weak cases were removed, and the final interpretation was based on human reading of the coded material. This reduces the risk of coding errors, but it does not remove that risk completely.

The thesis also follows basic internet-research ethics. Although the data was public, the topic can involve sensitive work situations, legal obligations, and employer or client relationships. For that reason, the analysis focuses on patterns across discussions rather than identifying or judging individual users. The thesis does not claim that a particular user violated an NDA or acted correctly. It only studies how developers publicly reason about confidentiality, disclosure, safeguards, and acceptable use.

4 Findings and Analysis

This section presents the findings from the coded developer discussions. The main evidence base is the final conservative strict subset: 113 coded responses from 43 public discussion threads. A coded response means one comment or answer that was analyzed using the coding schema.

The section is organized around the two research questions. The first part answers RQ1 by showing what developers treated as confidential information and what they understood as disclosure or exposure when AI-assisted coding tools were used under NDA-like constraints. The second part answers RQ2 by showing how developers defined acceptable use when project work was covered by NDAs, client confidentiality rules, company policy, or similar restrictions. The final part compares these findings with the broader high-signal conservative subset.

4.1 Overview of the Main Evidence Base

In the tables in this section, each count refers to coded responses, not individual developers. Percentages are calculated from the 113 responses in the main evidence base. Some codes can appear together in the same response, so some percentages do not add up to 100%.

Most responses were connected to both research questions. As shown in Table 3, 81 responses were coded as related to both RQ1 and RQ2. This matters because developers often discussed confidentiality and acceptable use together. They did not usually separate the question of what information is confidential from the question of whether AI tools can be used safely.

The dimension codes show the same pattern. `D2`, which covers disclosure, policy, retention, compliance, and vendor-risk reasoning, appeared most often. This shows that developers mainly focused on what happens when project information enters an AI tool, especially when the tool is connected to an external provider. `D1`, `D4`, and `D3` also appeared often, which shows that developers connected disclosure risk with confidential material, acceptable use, and safeguards.

Coding output in the main evidence base	**Coded responses**	**% of subset**
RQ coded as Both	81	71.7
RQ coded as RQ1 only	14	12.4
RQ coded as RQ2 only	18	15.9
D2 present	105	92.9
D1 present	72	63.7
D4 present	65	57.5
D3 present	42	37.2

Table 3: Research-question and dimension coding in the main evidence base.

The source mix is important for interpreting the findings. In the main evidence base, 76 responses came from Hacker News, 29 from Discourse, and 8 from GitHub Issues. The results therefore mainly reflect public discussions in technical communities, especially communities where developers

discuss software tools, security, and work practices. This overview gives the starting point for the rest of the findings. Developers in the corpus did not discuss AI-assisted coding only as a way to write code faster. They also discussed whether using these tools could conflict with NDAs, client confidentiality duties, company rules, or similar restrictions. The main concern was what happens to project information when it is entered into an AI tool: where it goes, who may access it, whether it is stored, what rules apply, and what safeguards are needed.

4.2 RQ1: Confidential Information and Disclosure

The first research question asks how developers interpret what counts as confidential information and disclosure when using AI-assisted coding tools under NDA-like or related confidentiality constraints. To answer this question, the analysis looked at two things in each coded response. First, it looked at what kind of project material the response treated as sensitive. Second, it looked at what kind of risk the response connected to AI-tool use. Together, these two parts show how developers interpreted the boundary between project information that should stay inside the work setting and information that might become exposed through an AI tool.

The main finding is that developers in the corpus did not treat confidentiality as only a question of source code. They treated it as a question of project context. Source code was part of the concern, but it was not the only concern. As shown in Table 4, credentials or secrets appeared more often than source code itself, and repository material, customer or client data, and prompts also appeared often. This matters because these categories are not separate from software work. They are the kinds of materials developers may use when asking an AI tool to debug, explain, review, or modify real project code.

Confidential artifact category	**Coded responses**	**% of subset**
Credentials or secrets	36	31.9
Repository or codebase	30	26.5
Source code	30	26.5
Customer or client data	28	24.8
Prompts or context window	16	14.2

Table 4: Most frequent confidential artifact categories in the main evidence base. Counts refer to coded responses out of 113.

The table shows that confidentiality was interpreted broadly. If developers were only worried about copying source code into an AI tool, source code would be the central category by itself. Instead, the coded responses show several kinds of project material appearing together. Credentials matter because they can give access to systems. Customer or client data matters because it can reveal business relationships or personal information. Repository context matters because it can show how a product is built. Prompts matter because they can combine small pieces of information into a description of a confidential task. This means developers often treated confidentiality as a boundary around project context, not only around complete source-code files.

The same pattern appears in how developers interpreted disclosure. Disclosure was not treated only as publishing information publicly. As shown in Table 5, the most frequent risk category was vendor access, followed by internal policy or compliance. This is important because it shows that developers were not only thinking about public leaks. They were also thinking about whether project information leaves the normal work setting and becomes available to an AI provider, a tool system, or another outside party.

Risk category	**Coded responses**	**% of subset**
Vendor access	79	69.9
Internal policy or compliance	66	58.4
Credential exposure	34	30.1
Customer privacy	29	25.7
Retention or training	28	24.8

Table 5: Most frequent disclosure and confidentiality risk categories in the main evidence base. Counts refer to coded responses out of 113.

The risk table shows that developers interpreted disclosure as a data-flow problem. The main question was not only whether information becomes public. It was also where the information goes, who can access it, whether it is stored, whether it can be reviewed, whether it may be used for training, and whether company or client rules allow that kind of processing. This is why vendor access and internal policy appear so strongly in the coded material. They show that developers were translating NDA-like duties into practical questions about tool providers, storage, retention, approval, and control.

This is where the NDA-related problem becomes visible in the data. Developers did not always use the word "NDA" directly. Instead, they often discussed NDA-like concerns through practical terms such as company policy, third-party access, vendor storage, retention, approval, privacy mode, and local deployment. This shows how broad NDA duties were translated into everyday development questions. Developers were asking whether project information can be shared with an AI tool, whether the tool provider counts as an outside party, and whether tool settings are enough to keep the work inside the allowed confidentiality boundary.

The answer to RQ1 is therefore that developers in the corpus interpreted confidential information as project context, not only as source code. They also interpreted disclosure as exposure of that context, not only as public release. Under NDA-like constraints, the central concern was whether AI-assisted coding moved project information outside the boundary set by the employer, client, or agreement.

4.3 RQ2: Acceptable Use, Safeguards, and Responsibility

The second research question asks how developers define acceptable use of AI-assisted coding tools in confidential work settings, including work covered by NDAs, client confidentiality duties, company policy, or similar restrictions. To answer this question, the analysis looked at three things

in each coded response. First, it looked at the position the response took toward AI-assisted coding, such as allowing use only with controls, accepting it more generally, rejecting it, or leaving the position unclear. Second, it looked at what safeguards were mentioned, such as approval, privacy settings, local deployment, enterprise tools, redaction, or avoiding sensitive material. Third, it looked at who was treated as responsible for safe use, such as the developer, employer, client, security team, or tool vendor.

The main finding is that developers in the corpus rarely treated acceptable use as a simple yes-or-no decision. Instead, they described acceptable use as conditional. AI-assisted coding became more acceptable when the exposure of project information was limited, approved, or controlled.

This finding appears most clearly in the norm-stance results. As shown in Table 6, the largest category was `allow_only_with_controls`. This category was much larger than `generally_accepting` or `reject_use`. This means that the dominant position was not unrestricted acceptance and not complete rejection. The most common position was that AI-assisted coding can be used in confidential work only when certain conditions are met.

Norm stance	**Coded responses**	**% of subset**
Allow only with controls	76	67.3
Generally accepting	17	15.0
Unclear	10	8.8
Reject use	8	7.1
Depends on company policy	2	1.8

Table 6: Norm stances about acceptable use in the main evidence base. Counts refer to coded responses out of 113.

The table shows that developers mainly normalized AI-assisted coding through conditions. They did not usually say that AI tools should simply be banned from confidential work. They also did not usually say that they could be used without concern. Instead, the largest group of responses framed acceptable use as something that depends on controls. In NDA-like settings, this means acceptable use was often connected to whether the developer could limit what information was exposed, whether the tool was approved, and whether company or client rules allowed the workflow.

The safeguards mentioned in the data show what these conditions looked like in practice. As shown in Table 7, approval or security review and local or self-hosted models were the most frequent concrete safeguards. Developers also discussed enterprise or privacy modes and avoiding use entirely. These categories show that developers often tried to make AI-assisted coding acceptable by changing the workflow around the tool. They discussed who approved the tool, where the model ran, what data was sent, and whether privacy settings reduced the risk.

The mitigation table also shows an important limit in the discussions. The most frequent mitigation category was `none_explicit`. This does not mean that developers ignored confidentiality risk. It means that many responses identified a risk without giving a concrete safeguard. In other words, developers were often able to say what the problem was, but not always how it should be managed.

This matters because it shows that acceptable use was not always supported by a clear practical rule.

Mitigation category	Coded responses	% of subset
None explicit	46	40.7
Approval or security review	38	33.6
Local or self-hosted model	23	20.4
Enterprise or privacy mode	11	9.7
Avoid use entirely	10	8.8

Table 7: Most frequent mitigation categories in the main evidence base. Counts refer to coded responses out of 113.

Developers often recognized that vendor access, retention, credentials, customer data, or company policy could create a problem. However, many responses did not explain whether the solution should be approval, local deployment, privacy mode, redaction, avoiding the tool, or something else. This suggests that the public discussions contained more risk recognition than settled guidance.

When developers did describe safeguards, the strongest pattern was organizational or infrastructural control. They discussed approval, security review, local deployment, enterprise settings, privacy modes, and avoiding sensitive material. This shows that acceptable use was often connected to changing the conditions around the tool, not simply trusting the developer to decide alone.

Responsibility was also spread across different actors. Tool vendors were mentioned most often, but developers, employers, clients, security teams, and unclear or shared responsibility also appeared in the data. This shows that developers did not describe confidentiality as only an individual developer's responsibility. They often discussed it as something shaped by several actors at once: the developer who chooses what to share, the employer or client who sets the rules, and the vendor who controls storage, retention, privacy settings, and access.

This matters for answering RQ2. Developers defined acceptable use not only by asking whether the tool was useful, but by asking whether responsibility and safeguards were clear. They described safer use through approval, security review, local deployment, privacy modes, redaction, and limiting the amount of project context shared with the tool. They described riskier use as tool use where project information was sent to an external service without clear approval, without control over retention, or without knowing who could access the data.

The answer to RQ2 is therefore that developers define acceptable use through conditions. They do not mainly present AI-assisted coding in confidential settings as always acceptable or always forbidden. Instead, they treat it as acceptable when the exposure of project information is controlled, the rules are clear, and responsibility is shared between developers, organizations, and tool providers.

4.4 Comparison with the Broader Dataset

The broader high-signal conservative subset was used as a comparison layer. The main findings come from the final conservative strict subset, because it has the clearest response-level evidence. The broader subset is used only to check whether the same patterns also appear when the inclusion boundary is wider.

The comparison supports the main result. In both datasets, developers discussed AI-assisted coding under NDA-like or related confidentiality constraints mainly as a question of disclosure and control. They were not only asking whether AI tools are useful. They were asking what happens when project information is entered into a tool, who may access it, whether it is stored, and whether company or client rules allow that use.

Table 8 shows that the same broad structure appears in both subsets. `D2`, which covers disclosure, policy, retention, compliance, and vendor-risk reasoning, is the strongest dimension in both datasets. `D1`, which covers confidential material and boundary definition, also remains strong. This supports RQ1: developers interpreted the issue through both confidential material and exposure risk. The broader subset therefore does not shift the finding toward a simple concern about source code alone. It keeps the focus on where project information goes and who may access it.

The comparison also supports RQ2. In both subsets, `allow_only_with_controls` remains the largest norm category. This shows that developers most often described AI-assisted coding as acceptable only when controls were in place. The broader subset is slightly more accepting, because `generally_accepting` appears more often there than in the strict subset. However, the main pattern remains conditional use rather than unrestricted acceptance.

Coding output	**Strict conservative %**	**High-signal conservative %**
RQ coded as Both	71.7	62.0
RQ coded as RQ1 only	12.4	14.0
RQ coded as RQ2 only	15.9	24.0
D2 present	92.9	89.5
D1 present	63.7	60.3
D4 present	57.5	55.5
D3 present	37.2	36.7
Conditionally permitted with controls	57.5	55.9
Broadly confidential	24.8	18.3
Allow only with controls	67.3	57.6
Generally accepting	15.0	23.1

Table 8: Comparison of coding patterns in the main evidence base and the broader comparison subset. The main evidence base is the strict conservative subset, which contains 113 coded responses from 43 public discussion threads. The broader high-signal conservative subset contains 229 coded responses from 71 threads. Percentages show the share of responses in each subset that received each code. Some categories can appear together in the same response, so percentages do not add up to 100%.

The table shows that broadening the dataset changes the tone slightly, but not the main interpretation. The broader subset contains more generally accepting responses, but it still centers on disclosure, policy, vendor access, and controlled use. This matters because it shows that the finding is not only an effect of using the strictest dataset. Across both subsets, developers frame AI-assisted coding under confidentiality constraints as a question of what information is exposed, what rules apply, what safeguards are needed, and who is responsible for acceptable use.

4.5 Summary of Findings

The findings show that developers in the corpus did not discuss AI-assisted coding under NDA-like or related confidentiality constraints as a simple yes-or-no issue. They discussed it as a question of controlled exposure. The central concern was what project information enters an AI tool, where that information goes, who may access it, and what controls are needed before that use becomes acceptable.

For RQ1, developers interpreted confidential information as broader than source code. They also treated credentials, repository context, client data, prompts, logs, documentation, and architecture details as potentially sensitive. They interpreted disclosure broadly as well. Disclosure did not only mean publishing information online. It also meant sending project information to an external AI service, exposing it to a vendor, storing it in a prompt history, or allowing an AI agent to inspect files in a workspace.

For RQ2, developers defined acceptable use through conditions. AI-assisted coding was usually treated as acceptable only when exposure was limited or controlled. The main controls discussed were approval, security review, local or enterprise tools, privacy settings, redaction, avoiding credentials or customer data, and limiting how much project context is shared. At the same time, many responses identified risks without giving clear safeguards, which shows that developers often recognized the confidentiality problem more clearly than they explained how to solve it.

The contribution of these findings is to show how developers translate NDA-like duties into practical AI-tool decisions. The thesis does not only show that developers are concerned about confidentiality. It shows how they reason through that concern: by identifying sensitive project material, asking whether AI-tool use counts as disclosure, comparing tool settings and deployment options, and discussing who should be responsible for acceptable use.

5 Discussion and Limitations

The findings show that the main issue is not simply whether developers use AI coding tools. The more important issue is how they decide what project information can safely enter those tools. This matters because AI-assisted coding often works better when developers provide real project context, but that same context may contain confidential information. A prompt, log, repository snippet, credential, or architecture description may help the tool produce a better answer, but it may also move protected information outside the employer or client boundary.

This is why the thesis adds to research on AI-assisted coding. Existing research shows that developers use these tools for coding, debugging, explanation, and problem solving [Barke et al., 2022, Zhang et al., 2023, Hao et al., 2024]. This thesis shows that, in confidential work settings, tool use is also shaped by questions about vendor access, storage, retention, company policy, approval, and safeguards. It also adds to confidentiality research by showing how broad NDA-like duties are interpreted in practical developer discussions, not only in legal or policy documents [Hrdy and Seaman, 2024].

The results have practical implications for developers, employers, clients, and tool providers.

Developers need clearer guidance because they are often the ones deciding what information to include in a prompt or workspace. Employers and clients need policies that say more than "do not share confidential information," because developers need to know how that rule applies to prompts, logs, repository context, privacy modes, local models, and approval. Tool providers matter because developers judge acceptable use partly through what the tool does with user input, including storage, retention, privacy settings, vendor access, and enterprise controls.

The study has limitations. The data comes from public online discussions, so it captures public reasoning rather than private workplace behavior. The corpus was built through targeted searches, not by asking a random group of developers. The searches were designed to find discussions where people were already talking about AI-assisted coding, confidentiality, NDAs, disclosure, safeguards, or acceptable use. For this reason, the findings show patterns in these collected public discussions. They do not show how common these views are among all developers. The coding was partly supported by an LLM, so the findings should be understood as structured qualitative interpretation rather than automated truth. Finally, the thesis does not decide whether a specific use of an AI tool would legally violate an NDA. It studies how developers discuss and interpret those boundaries.

6 Conclusion

This thesis studied how developers publicly reason about AI-assisted coding under NDA-like confidentiality constraints. The aim was not to decide whether particular legal claims made online were correct. The aim was to understand how developers interpret confidentiality, disclosure, safeguards, and acceptable use when AI coding tools become part of practical software work.

The findings answer RQ1 by showing what developers in the corpus treated as confidential. The analysis coded the project materials mentioned in the discussions. Source code appeared as one sensitive material, but it was not the only one. Developers also discussed credentials, private repositories, client data, logs, prompts, documentation, and architecture details. This means that, in these discussions, the confidentiality problem was not only about pasting a full code file into an AI tool. It was also about smaller pieces of project information that could reveal something about the system, the client, or the organization.

The findings also show how developers in the corpus understood disclosure. They did not discuss disclosure only as publishing information on a public website. They also discussed it as sending project information to an AI provider, storing it in a chat history, allowing it to be reviewed, using it for model improvement, or letting an AI coding agent read files in a workspace. This connects directly to NDA-like rules because NDAs often restrict disclosure to third parties and unauthorized use of confidential information. The discussions show that developers were trying to understand whether using an AI tool could cross that boundary, even when the information was not made public.

The findings answer RQ2 by showing when developers in the corpus treated AI-assisted coding as acceptable in confidential work. They did not usually present one simple answer, such as "always allowed" or "always forbidden." Instead, they connected acceptable use to specific conditions. AI-assisted coding was treated as more acceptable when the developer removed secrets, avoided client data, used a local or approved tool, enabled privacy settings, followed company policy, or

asked for approval. It was treated as less acceptable when project information was sent to an external AI service without clear permission or without knowing how the provider would store, review, or use the data.

The main result is that developers discussed AI-assisted coding under NDA-like rules as a boundary problem. They asked what information is protected, where that information goes when an AI tool is used, who is allowed to receive it, and what safeguards are needed before the tool can be used. This is the gap the thesis fills: it shows how developers themselves reason through NDA-related confidentiality questions in public technical discussions.

Future work could build on this study by comparing public discussions with interviews, company policies, or legal analysis of AI-tool use under confidentiality agreements. This would help show how public reasoning, workplace rules, and legal interpretation align or differ.

References

[Al Alamin et al., 2021] Al Alamin, M. A., Malakar, S., Uddin, G., Afroz, S., Haider, T. B., and Iqbal, A. (2021). An empirical study of developer discussions on low-code software development challenges. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*.

[Anthropic, 2026] Anthropic (2026). Privacy policy.

[Anysphere, 2025] Anysphere (2025). Cursor Privacy Policy. Last updated October 6, 2025.

[Barke et al., 2022] Barke, S., James, M. B., and Polikarpova, N. (2022). Grounded copilot: How programmers interact with code-generating models. arXiv preprint.

[Butler et al., 2024] Butler, J., Suh, J., Haniyur, S., and Hadley, C. (2024). Dear diary: A randomized controlled trial of generative ai coding tools in the workplace. arXiv preprint.

[Chew et al., 2023] Chew, R., Bollenbacher, J., Wenger, M., Speer, J., and Kim, A. (2023). Llm-assisted content analysis: Using large language models to support deductive coding.

[Croft et al., 2021] Croft, R., Xie, Y., Zahedi, M., Babar, M. A., and Treude, C. (2021). An empirical study of developers' discussions about security challenges of different programming languages. arXiv preprint.

[Delile et al., 2023] Delile, Z., Radel, S., Godinez, J., Engstrom, G., Brucker, T., Young, K., and Ghanavati, S. (2023). Evaluating privacy questions from Stack Overflow: Can ChatGPT compete? In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*.

[El Aoun et al., 2022] El Aoun, M. R., Li, H., Khomh, F., and Openja, M. (2022). Understanding quantum software engineering challenges: An empirical study on Stack Exchange forums and GitHub issues. arXiv preprint.

[Franzke et al., 2020] Franzke, A. S., Bechmann, A., Zimmer, M., Ess, C., and the Association of Internet Researchers (2020). Internet research: Ethical guidelines 3.0.

[GitHub, 2022] GitHub (2022). Github copilot is generally available to all developers. Accessed 20 June 2026.

[GitHub, 2026] GitHub (2026). GitHub Copilot Trust Center.

[Grimmer et al., 2022] Grimmer, J., Roberts, M. E., and Stewart, B. M. (2022). *Text as Data: A New Framework for Machine Learning and the Social Sciences*. Princeton University Press.

[Hao et al., 2024] Hao, H., Hasan, K. A., Qin, H., Macedo, M., Tian, Y., Ding, S. H. H., and Hassan, A. E. (2024). An empirical study on developers shared conversations with ChatGPT in GitHub pull requests and issues.

[Horstmann et al., 2025] Horstmann, S. A., Hong, S., Klein, D., Serafini, R., Degeling, M., Johns, M., Moonsamy, V., and Naiakshina, A. (2025). "sorry for bugging you so much." exploring developers' behavior towards privacy-compliant implementation. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1159–1177.

[Hrdy and Seaman, 2024] Hrdy, C. A. and Seaman, C. B. (2024). Beyond trade secrecy: Confidentiality agreements that act like noncompetes. *Yale Law Journal*, 133(3):669–754.

[Hsieh and Shannon, 2005] Hsieh, H.-F. and Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research*, 15(9):1277–1288.

[Kozinets, 2019] Kozinets, R. V. (2019). *Netnography: The Essential Guide to Qualitative Social Media Research*. SAGE Publications.

[National Institute of Standards and Technology, 2024] National Institute of Standards and Technology (2024). Artificial intelligence risk management framework: Generative artificial intelligence profile. NIST AI 600-1.

[Nissenbaum, 2004] Nissenbaum, H. (2004). Privacy as contextual integrity. *Washington Law Review*, 79:119–158.

[Oksanen et al., 2025] Oksanen, J., Lucero, A., and Hämäläinen, P. (2025). Llmcode: Evaluating and enhancing researcher-ai alignment in qualitative analysis. arXiv preprint.

[OpenAI, 2022] OpenAI (2022). Introducing chatgpt. Accessed 20 June 2026.

[OpenAI, 2026a] OpenAI (2026a). Data controls faq. Accessed 20 June 2026.

[OpenAI, 2026b] OpenAI (2026b). Structured model outputs. Accessed 25 June 2026.

[OpenAI, 2026c] OpenAI (2026c). US Privacy Policy. Updated May 18, 2026.

[OWASP Foundation, 2025] OWASP Foundation (2025). Owasp top 10 for llm applications 2025.

[Shrivastava et al., 2022] Shrivastava, D., Larochelle, H., and Tarlow, D. (2022). Repository-level prompt generation for large language models of code. arXiv preprint.

[Tahaei et al., 2023] Tahaei, M., Abu-Salma, R., and Rashid, A. (2023). Stuck in the permissions with you: Developer and end-user perspectives on app permissions and their privacy ramifications. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM.

[Tahaei et al., 2022] Tahaei, M., Vaniea, K., and Rashid, A. (2022). Embedding privacy into design through software developers: Challenges and solutions. arXiv preprint.

[UK Intellectual Property Office, 2015] UK Intellectual Property Office (2015). Non-disclosure agreements. Accessed 20 June 2026.

[Ulfsnes et al., 2024] Ulfsnes, R., Moe, N. B., Stray, V., and Skarpen, M. (2024). Transforming software development with generative ai: Empirical insights on collaboration and workflow. arXiv preprint.

[Xiao et al., 2023] Xiao, Z., Yuan, X., Liao, Q. V., Abdelghani, R., and Oudeyer, P.-Y. (2023). Supporting qualitative analysis with large language models: Combining codebook with gpt-3 for deductive coding. arXiv preprint.

[Zhang et al., 2023] Zhang, B., Liang, P., Zhou, X., Ahmad, A., and Waseem, M. (2023). Practices and challenges of using GitHub copilot: An empirical study.

[Zhu et al., 2022] Zhu, W., Zhang, H., Hassan, A. E., and Godfrey, M. W. (2022). An empirical study of question discussions on Stack Overflow. *Empirical Software Engineering*, 27(6).

A First-Stage Lexical Filtering Codebook

The first relevance filter used the following predefined keyword categories. Matching was case-insensitive. A category added its points once when at least one term from that category appeared. Multiple terms from the same category did not add extra points.

AI-tool language (+3):

`chatgpt; openai; copilot; github copilot; copilot chat; claude; claude code; cursor; gemini; gemini code assist; codewhisperer; amazon q; amazon q developer; cody; sourcegraph cody; tabnine; codex; llm; llms; large language model; generative ai; ai assistant; ai tool; ai tools; ai coding assistant; ai code assistant; code generation; prompt engineering.`

Confidentiality language (+3):

`nda; ndas; non disclosure agreement; non-disclosure agreement; nondisclosure agreement; confidential; confidentiality; proprietary; trade secret; trade secrets; internal code; private repo; private repository; client code; customer code; company secret; source available only internally; sensitive code; private code; work code; company code; proprieta information; sensitive information; sensitive data; company data; work data; internal docs; internal documentation; customer data; production data; ip; intellectual property.`

Code language (+2):

`code; source code; codebase; repository; repo; snippet; prompt; stack trace; schema; sql; sdk; pull request; commit; branch; api key; token; secret; config file; env file; credentials; database; database schema; terraform; yaml.`

Work-context language (+2):

`work; workplace; employer; company; corporate; enterprise; client; customer; internal; production; policy; security team; legal; compliance; manager.`

Boundary or disclosure language (+2):

`paste; share; send; upload; submit; copy; disclose; disclosure; prompt; feed; attach; include; enter; leak; expose; retain; store; log; logging; training; train; access; review.`

Legal or authorization language (+2):

`violate; violation; enforceable; enforcement; breach; contract; agreement; lawyer; legal risk; policy violation; allowed; not allowed; permitted; permission; not permitted; approved; approval; ban; banned; forbidden.`

Mitigation language (+2):

`sanitize; sanitized; anonymize; anonymized; redact; redacted; abstract; abstraction; pseudocode; minimal snippet; small snippet; strip identifiers; remove names; mask; masked; obfuscate; obfuscated; dummy data; fake data; generalize; generalized; summarize.`

Normative language (+1):

everyone does it; nobody cares; normal; common practice; best practice; should; should not; acceptable; unacceptable; fine; not fine; risky; safe; okay; not okay; what do you do; what do you all do.

Engineer-discourse language (+1):

developer; engineer; programmer; software; backend; frontend; devops; repository; code review; production.

Two structural signals also added one point each:

- the response contained at least 120 characters;
- One additional point was given when the record represented a user's response to a thread, rather than the thread title or original post.

Generic exclusion terms:

privacy policy; terms of service; advertisement; sponsored; newsletter; subscribe; meeting notes; release notes; generated summary.

GitHub Issues exclusion terms:

automatically locked; stale bot; duplicate issue; reply to this email directly; this issue has been inactive.

DEV.to exclusion terms:

follow me for more; affiliate; sponsor.

B Second-Stage Selection Keywords

The high-signal and strict corpora were two parallel selections from the broad first-filter corpus. The following keyword categories were used during these selections.

AI terms checked in titles or parent questions:

chatgpt; copilot; claude; cursor; codex; openai; gemini; llm; mcp; ai.

Confidentiality and scope terms checked in titles:

nda; confidential; privacy; private; internal; company; customer; client; repo; repository; codebase; source code; file; files; upload; uploaded; knowledge; document; docs; security; compliance; policy; permission; approval; approved; allowed; forbidden; retain; retention; store; stored; server; training; logging; local mode; privacy mode; exclude; cursorignore; embedding; embeddings; project-scoped; developer mode.

Risk and policy terms checked in parent questions:

nda; confidential; confidentiality; privacy; private; proprietary; internal; company; customer; client; security; compliance; policy; legal; approved; approval; allowed; forbidden; retain; retention; store; stored; server; training; log; logging.

Boundary terms checked in parent questions:

repo; repository; codebase; source code; snippet; file; files; upload; uploaded; knowledge; document; docs; context; prompt; embedding; embeddings; project-scoped; mcp server; cursorignore; privacy mode; local mode.

Evidence terms checked in individual responses:

confidential; privacy; private; internal; company; customer; client; repo; repository; codebase; source code; file; files; upload; uploaded; knowledge; document; docs; context; prompt; server; store; stored; retain; retention; training; logging; policy; security; compliance; approved; approval; allowed; forbidden; sanitize; redact; exclude; ignore; cursorignore; privacy mode; local mode.

High-signal source-specific terms

The high-signal filter also used platform-specific rules because the platforms contained different types of discussions and noise.

Discourse inclusion terms:

privacy; private; file; files; upload; uploaded; knowledge; repo; repository; codebase; security; compliance; policy; permission; retain; retention; local mode; privacy mode; exclude; cursorignore; developer mode; doc; docs; context; memory; project; workflow; rule; rules; review; tool; embedding; index; server; enterprise; redaction; mcp; codex.

GitHub Issues inclusion terms:

use of ai tools; 0.6 use of ai tools; sanitized; policy.

Reddit inclusion terms:

code; assessment; policy; security; private; privacy; work; server.

Stack Exchange title terms:

claude code; ai chat; gpt; mcp.

For Hacker News, clear AI and confidentiality context in the parent discussion could support a response even when the response itself did not contain one of the response-evidence terms.

Source-specific exclusion phrases

Discourse exclusions:

games inc.; soul framing; strategic advice; prompt engineering is dead; dangerous lack of transparency and informed consent; ai pulse edition; pricing; launches; what are you building; share your projects; billing; token consumption; vision cost; too slow; recursive 'nonsense'; co-founder; hackathon; developer community; how to improve gpt; how to rewrite prompts; another lag-victim; hot take about cursor; chatgpt has been my friend; need a change and way to use my new skills.

Hacker News exclusions:

`dependency health checker; job descriptions; skeptic impressed; static analysis toolkit.`

GitHub Issues strict-title exclusions:

`burning tokens very fast; removing the agents.md file again; gen ai - llm, rag, langchain; rfc: skills catalog; taxonomy validation; model degradation; extensions fails to install; pandora's box; bounty; news; regulation.`

Stack Exchange strict-title exclusions:

`authenticate into apache; pointerlockcontrols; vitetest; webflow using socket.io; odata; allow the mouse; validationexception.`

C Complete Query Bank

The query bank was defined before data collection and operationalized the four analytical dimensions used in this study. It contained 15 topic groups with five alternative search phrases in each group, resulting in 75 queries.

Groups 1–13 focused primarily on one analytical dimension. Groups 14 and 15 covered issues that crossed two dimensions. The group titles below provide a plain-language explanation of each search topic, while the queries are reproduced exactly as they appeared in the collection configuration.

Dimension	Groups	Primary search focus
D1	1–4	Confidential material and actions that may disclose it
D2	5–8	NDA risk, policy, vendor practices, and enterprise controls
D3	9–11	Sanitization, minimization, and safe workflows
D4	12–13	Workplace norms, acceptable use, and team rules
D2/D4	14	Retention and training as risk and acceptability issues
D1/D2	15	Product-specific confidential material and vendor risk

Table 9: Organization of the query groups across the analytical dimensions.

C.1 D1: Confidential Material and Disclosure Actions

Group 1: Defining confidential or sensitive code

1. `what counts as confidential code in chatgpt`
2. `is internal code confidential when using ai assistant`
3. `can i share client code with copilot`
4. `what parts of source code are safe to paste into llm`
5. `what is considered sensitive code for ai tools`

Group 2: Actions that may disclose project information

1. `can i paste company code into chatgpt`
2. `share stack trace with chatgpt company code`
3. `upload proprietary code to ai assistant`
4. `paste internal repository snippet into claude`
5. `can i send work code to cursor`

Group 3: Private repositories and customer or client data

1. `private repo ai assistant confidential`
2. `customer data in ai coding assistant`
3. `client code chatgpt confidential`
4. `work code claude code`
5. `company repository cursor confidential`

Group 4: Logs, credentials, configuration, and schemas

1. `stack trace chatgpt confidential`
2. `api keys in copilot prompt`
3. `database schema chatgpt company policy`
4. `config file claude confidential`
5. `production logs llm confidential`

C.2 D2: Legal, Policy, and Vendor Risk

Group 5: NDA violations and legal risk

1. `does using copilot violate nda`
2. `is this nda violation chatgpt code`
3. `legal risk of pasting confidential code into llm`
4. `non disclosure agreement ai coding assistant`
5. `breach nda by using cursor with company code`

Group 6: Company policy, compliance, and approval

1. `company policy ai coding assistant confidential`
2. `security policy chatgpt prompt company code`
3. `approved use of ai assistant with source code at work`
4. `work policy for copilot confidential code`
5. `legal approval to use claude code at work`

Group 7: Vendor storage, retention, and model training

1. `does cursor store company code`
2. `does github copilot train on private repo`
3. `claude code retention company code`
4. `gemini code assist private code policy`
5. `sourcegraph cody private code training`

Group 8: Enterprise security and governance controls

1. `enterprise ai coding assistant confidential data`
2. `security review for ai coding tools work`
3. `can legal ban chatgpt for source code`
4. `compliance requirements for github copilot company`
5. `using ai tools with customer code policy`

C.3 D3: Safeguards and Mitigation

Group 9: Sanitization, anonymization, and redaction

1. `sanitize code snippet before chatgpt`
2. `anonymize internal code before copilot`
3. `redact proprietary identifiers before llm`
4. `pseudocode instead of source code llm`
5. `obfuscate company code before ai assistant`

Group 10: Minimizing the code and context shared

1. `minimal snippet chatgpt policy`
2. `small code snippet safe for chatgpt`
3. `abstract company code before ai assistant`
4. `share only function signature with llm`
5. `use fake data in prompt for work code`

Group 11: Safe workflows for confidential code

1. `how to use chatgpt safely with company code`
2. `safe workflow for copilot and confidential code`
3. `how developers sanitize prompts for work code`
4. `best way to redact code before claude`
5. `mask identifiers before using cursor ai`

C.4 D4: Norms and Acceptable Use

Group 12: Workplace norms and developer expectations

1. `is everyone using chatgpt with work code`
2. `normal to use copilot with company repository`
3. `best practice ai coding assistant confidential code`
4. `team norms around chatgpt and internal code`
5. `do developers use cursor with company code`

Group 13: Acceptable use and team rules

1. `is it okay to use claude code at work`
2. `what do you tell your team about chatgpt and source code`
3. `acceptable use of ai coding tools with internal code`
4. `do engineers paste work code into llms`
5. `team rules for github copilot confidential code`

C.5 Cross-Cutting Query Groups

Group 14: Logging, retention, and training (D2/D4)

1. `does chatgpt store proprietary code`
2. `llm prompt retention confidential source code`
3. `ai coding assistant training on internal code`
4. `does copilot retain private repository code`
5. `does cursor log prompts with company code`

Group 15: Product-specific confidentiality risks (D1/D2)

1. `github copilot private repository confidential`
2. `claude code company code nda`
3. `cursor company policy confidential code`
4. `amazon q developer proprietary code`
5. `tabnine private code policy`