Knowledge Reuse in ML Prototyping: Insights from an Interview Study

Selin Coban
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
e-mail: coban@swc.rwth-aachen.de

Patrick Chrestin
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
e-mail: patrick.chrestin@rwth-aachen.de

Horst Lichter
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
e-mail: lichter@swc.rwth-aachen.de

***Abstract*—Prototyping is a core activity in Machine Learning (ML) solution development, yet research on reuse in this context has largely focused on source code, overlooking more comprehensive forms of knowledge that ML developers rely on. This paper reports on a qualitative interview study with 18 ML developers from academia and industry that investigates how knowledge is sought, evaluated, reused, and retained during ML prototyping. Our thematic analysis reveals that ML developers reuse not only code but also declarative knowledge (e.g., terminology and foundational concepts) and procedural knowledge (e.g., design patterns and instructions). Search strategies are opportunistic and context-driven. Reuse decisions are based on quality, intuition, and direct experimentation. Notably, retention practices are often unsystematic, relying on memory or unstructured notes, which hinders the effective transfer of knowledge across projects. With this paper, we contribute a more structured perspective on knowledge reuse in ML prototyping. We further highlight the need for dedicated tools to support knowledge management in ML solution prototyping and suggest directions for systematically retaining this knowledge to increase its reuse.**

Keywords-Knowledge Reuse; Prototyping; Machine Learning.

I. Introduction

Prototyping is a central practice in software engineering, enabling developers to explore alternatives and mitigate risk. In Machine Learning (ML), however, prototyping serves a distinct purpose. Rather than emphasizing architectural or implementation decisions, ML prototyping is driven by rapid experimentation with data, algorithms, and models to assess the feasibility and potential value of a solution before committing to product development.

ML solution prototyping is inherently exploratory. When addressing a specific problem, practitioners navigate the solution space by examining related problems and leveraging existing solutions, reusing ideas, approaches, and artifacts throughout the process. Although code reuse in ML prototyping has been studied, considerably less is known about how practitioners reuse more comprehensive forms of knowledge, such as methodological choices or evaluation strategies.

Reuse is an essential means of developing more efficiently. Reuse in software development is generally defined as a process guided by a specific strategy that specifies what can be reused and how [1]. This strategy dictates the development and management of software artifacts to ensure they are easily accessible for reuse. The reuse process must be adapted to its context, as reuse activities must also be integrated with development activities. Dedicated tools for reuse activities are required to quickly and easily retrieve and apply reusable artifacts.

Without a clear understanding of how knowledge is reused during ML solution prototyping, valuable insights generated in this phase risk remaining fragmented, inconsistently retained, or lost altogether, limiting opportunities for reuse across projects. A deeper understanding of ML developers' knowledge-reuse practices can inform the design of processes and tools that better support ML work, reduce duplication of effort, and enable more systematic exploration of alternative solutions. To address this gap, we conducted and analyzed an interview study of ML developers focused on knowledge reuse during prototyping.

The remainder of this paper is structured as follows. Section II summarizes the related work on knowledge reuse. We formulate the research questions in Section III. Section IV describes the study method used to answer these research questions. Section V presents the results of the study for each question. In Section VI, we discuss these results and formulate some conclusions for improved prototyping tool support. Section VII outlines the threats to validity. Finally, Section VIII concludes the paper and suggests directions for future research.

II. Related Work

This section reviews existing research on knowledge reuse in software engineering and ML solution prototyping. It further briefly introduces the Information Foraging Theory, which explains how people search for and select information.

A. Knowledge Reuse in Software Engineering

In knowledge management, a distinction is made between *explicit* knowledge, which is documented and readily shareable, and *tacit* knowledge, which encompasses experiential know-how [2]. In software engineering, knowledge is commonly categorized into four types: organizational, managerial, technical, and domain-specific. Such knowledge may be embedded within software artifacts, for example, domain models, or exist solely in the minds of developers, as in the case of design decisions.

Knowledge reuse has long been studied in software engineering, with most work focusing on source code and model reuse, often supported by recommender systems [3]–[5]. Adjandra et al. conducted a systematic literature review to provide an overview of the challenges of knowledge reuse

in software engineering. They identified insufficient documentation, knowledge silos, and the inherent complexity of knowledge management as key factors hindering efficient reuse [6]. Exploratory empirical studies, such as [7], show that source code and design patterns are among the most frequently reused artifacts, and that the quality of artifact documentation strongly influences the likelihood of reuse. Source code is often copied and reused across projects, a practice that can negatively impact software quality [8].

Stack Overflow has been studied extensively as a source of code reuse, particularly with respect to reuse behavior and recommender support [9]–[12].

B. Knowledge Reuse in ML Solution Prototyping

With regards to ML prototyping, previous studies have identified widespread source code reuse in Jupyter Notebooks (notebooks short), often in the form of source code cloning [13]–[16], particularly from sources such as Stack Overflow [17] and GitHub [18]. Researchers have also examined the types of activities supported by such source code reuse, with data visualization being among the most common [19].

Since reusing source code is an essential activity during prototype development, researchers have also investigated how tools integrated into notebooks can support this activity by retrieving relevant code more efficiently than manual searches. NBSEARCH supports semantic code search in notebooks [20]. JUPYSIM helps developers retrieve similar notebooks based on user queries, but constructing complex graph-based queries may hinder adoption [21]. ELYRA enables manual storage and management of source code snippets extracted from notebooks [22]. A fully automatic recommendation approach is presented in TYPHON, which primarily relies on the similarity of markdown text cells rather than code cells [23]. For data analysis and exploration, PYSNIPPET recommends code snippets from ML library documentation and Stack Overflow [24], while EDAASSISTANT offers code search and recommendation functionality [25]. Recently, Large Language Models (LLMs) have been explored to enable semantic search notebooks [26].

Regarding ML model reuse, Peixoto et al. [27] propose a recommendation approach for ML models based on data similarity. AutoML techniques aim to automate the process of selecting and configuring ML models for a given task. Some AutoML approaches, particularly those using meta-learning, leverage knowledge from previous ML problems to guide model selection and hyperparameter optimization [28].

Further best practices supporting source code reuse include ensuring the reproducibility of notebooks, sharing data along with notebooks, and publishing ML libraries [15][29]–[31].

C. Information Foraging Theory

The Information Foraging Theory (IFT) explains how people seek information by balancing its expected value against the cost of acquiring it [32]. According to IFT, users aim to maximize information gain while minimizing time and cognitive effort, particularly in complex and uncertain environments, such as the Internet.

A key concept in IFT is information scent, which refers to cues that signal the potential usefulness of an information source, such as titles or images on a webpage. Users rely on these cues to quickly decide which sources to explore further and which to abandon.

IFT also characterizes information-seeking behavior as opportunistic and iterative. Users refine their queries iteratively, switch between sources, and adjust their goals based on intermediate results. Hereby, users often favor shallow exploration of multiple sources over deep analysis of a few, especially when time is constrained. In this paper, IFT is used as additional context to interpret the findings of our interview study.

III. RESEARCH QUESTIONS

Despite its importance, knowledge reuse in ML prototyping remains insufficiently understood. Studies of Jupyter Notebook practices have highlighted extensive code cloning but have paid little attention to the broader spectrum of knowledge artifacts and to the strategies practitioners use to search for, evaluate, and retain them. As a result, we lack a comprehensive understanding of how ML developers actually engage in knowledge reuse during prototyping.

The review of publications related to the knowledge reuse in ML prototyping provided the basis for the following research questions:

- **RQ1**: What types of knowledge are reused?
- **RQ2**: How do ML developers search for knowledge?
- **RQ3**: Where do ML developers search for knowledge, and what tools do they use to do so?
- **RQ4**: How do ML developers evaluate knowledge and its sources to decide whether they want to reuse them?
- **RQ5**: How do ML developers retain relevant knowledge?

Overall, by answering these research questions, we aim to provide a holistic conceptual view of knowledge reuse in ML solution prototyping, focusing on the type of knowledge reused (RQ1), the reuse process (RQ2, RQ3, RQ5), and the decision-making mechanisms underlying reuse (RQ4).

This paper addresses these questions, providing a basis for examining the full range of knowledge types reused by ML developers and their strategies for searching, evaluating, and retaining knowledge.

IV. STUDY METHOD

We opted for Cognitive Task Analysis (CTA) [33], in which participants verbalize their reasoning as they mentally walk through a task guided by semi-structured interview questions. Participants were presented with a concrete ML solution prototyping scenario to ensure that their responses were grounded in actual practice. Our goal was to understand why ML developers make specific reuse decisions during prototyping, insights that cannot be reliably captured through observation alone. Many reuse decisions stem from internal cognitive processes, such as recalling prior solutions, evaluating alternatives, and weighing trade-offs, which are not directly observable.

To this end, we conducted semi-structured interviews to collect qualitative data, followed by a thematic analysis of the interview transcripts to generate both qualitative insights and quantitative observations. This approach enables us to examine not only the knowledge and practices reported, but also their frequency of occurrence across participants.

A. Interview Design and Execution

Participants were asked how they would approach the following simple and abstract ML classification task:

> You are given the map of a public building. The map contains rooms, hallways, the library, the dining room, as well as all further offices. You are also given an extensive dataset of walking patterns, that are classified in GROUP-1, GROUP-2, GROUP-3, and OTHER. Your task now is to train a model that predicts the type of group when given a walking pattern.

We selected this task to evaluate not technical proficiency or domain knowledge, but the reasoning processes underlying knowledge search and reuse. With this task as a foundation, the interview questions were closely aligned with our research questions.

Before the main study, we conducted a pilot interview to refine the protocol and estimate its expected duration. In total, we interviewed 18 participants from both academic and industry contexts. An overview of the demographic information is presented in Table I. Participants were recruited through professional networks to ensure heterogeneity in disciplinary backgrounds and expertise in software engineering and machine learning. All participants reported having experience with ML solution prototyping.

TABLE I. INTERVIEW PARTICIPANTS BY POSITION TYPE AND DOMAIN.

ID	Role	Domain	ML Exp. (yrs)	Org. Size
Industry Practitioners				
1	Software Developer	IT Consulting and Software Dev.	3	<250
2[A]	ML Practitioner	Public Transport	4.5	<50
3[A]	ML Practitioner	Public Transport	9	<50
4	Student Worker	Information Processing	2	<50
5	Data Analyst	Finance	1	1
8	Data Owner	Clothing Retail	3	>10k
10	Software Developer	Real Estate	2	<250
12	Consultant	Corporate Consulting	4	<50
14	Software Developer	Process Automation	3	<50
15	Software Developer	AI Consulting	4.5	<50
16	AI Expert	Optimization Software	1	<5k
18	Consultant	Green Energy	1.5	<5k
Academic Practitioners				
6	Ph.D. Student	Manufacturing	3	<10
7	Student	Computer Science	2	N/A
9	Ph.D. Student	Materials Engineering	5	<100
11	Ph.D. Student	Medicine	0.5	<25
13	Researcher	Mechanical Engineering	1	N/A
17*	Ph.D. Student	Computer Vision	6	<25

* Interview split into two parts due to interviewee availability. IDs assigned in interview order; same letter indicates participants from the same organization. N/A means that the organization size for these participants is unknown.

The interviews were conducted by one researcher via videoconferencing between December 2024 and March 2025. All participants were informed about the purpose of the interview and the procedures for handling data. Each interview lasted between 32 and 87 minutes. All interviews were audio-recorded with participant consent, transcribed verbatim, pseudonymized, and stored for analysis.

B. Interview Analysis

To analyze the transcripts, we applied Thematic Analysis (TA) according to Braun and Clarke [34], using an inductive and semantic coding approach. This choice was driven by the exploratory nature of our study, which aimed to understand how ML developers search for and reuse knowledge.

Two researchers, referred to as the "TA Team," carried out the coding and theme development. In line with TA, codes were generated directly from the data, reflecting participants' explicit statements. Through discussion and refinement, the TA team organized these codes into conceptual categories, which it then developed into themes. The TA team determined the frequency with which each code and topic appeared in the interviews.

The TA process involved the following steps:

- *Data familiarization:* The TA team immersed themselves in the transcripts to gain a comprehensive understanding of the content.
- *Initial coding cycle:* Next, the TA team independently conducted open, inductive coding using the qualitative analysis tool *Delve* [35]. This step involved identifying and labeling meaningful data segments related to the research questions. Codes were kept descriptive and grounded closely in the participants' language, minimizing interpretive bias and preserving original intent.
- *Consensus and second coding cycle:* After the initial coding cycle, the TA team compared results, discussed differences, and reached consensus on the codebook. The codebook was then systematically applied to all transcripts in a second coding cycle to ensure consistency and analytical rigor.
- *Theme development:* The TA team organized the codes into categories, serving as candidate themes. These candidate themes were iteratively refined and aligned with the research questions.
- *Theme refinement and naming:* The TA team refined the themes through collaborative discussions to ensure clarity. Each theme was named to reflect its essence.
- *Reporting:* In this paper, we report on the results achieved. We excluded all codes that appeared in fewer than four interviews (less than 20% of the total), except where a code provided significant interpretive value.

Across the 18 interviews, we identified 89 unique codes. Only ten appeared in fewer than four interviews, and only four were mentioned a single time. This distribution indicates that most practices occurred across participants, suggesting that thematic saturation was achieved primarily. Additional interviews would likely have revealed only minor or context-specific practices rather than new overarching themes.

C. Interview Material

We provide the complete dataset, including themes, codes, quotes, their mappings, interview questions, full transcripts, and a code-by-code occurrence matrix as supplementary material on Zenodo [36].

V. Results

This section presents all identified themes and subthemes, organized by research question. Each theme is labeled (e.g., T1 for theme 1); participants are labeled similarly (e.g., P1 refers to participant 1).

A. What types of knowledge are reused (RQ1)?

Established concepts in knowledge management guided the coding and categorization of identified knowledge into broader themes. Our analysis revealed that ML developers actively search for *explicit* knowledge. We distinguish three partially interrelated categories: *declarative*, *procedural*, and *executable* knowledge. We use the term *knowledge element* to refer to explicit, codified units of knowledge that can be stored, communicated, and reused. Based on these categories, Table II provides an overview of the types of knowledge elements mentioned by ML developers.

T1 Declarative Knowledge: It refers to factual and conceptual knowledge that ML developers use to understand the problem domain or to frame their solution approach [37]. Seven participants searched for relevant *terminology* within the problem domain to further refine their search queries. They also explored *foundations*, such as commonly used metrics or benchmarks, which they considered necessary for evaluating the relevance and quality of reusable solutions. 12 ML developers searched for *ML algorithms* to determine which ones could be useful for the given problem. Although algorithms inherently involve procedures, participants typically sought high-level overviews and conceptual distinctions (e.g., when to use a random forest), rather than implementation details.

T2 Procedural Knowledge: It refers to knowledge that combines conceptual understanding with guidance on how to act [37]. 11 participants searched for a *solution design pattern*, meaning a general sequence of steps required to achieve a specific goal, including data preprocessing and feature engineering. Others explicitly sought *step-by-step instructions* that combine solution steps with practical guidance, such as code snippets or tutorials, directly applicable to the given problem. Both guide ML developers in applying their knowledge and skills to solve the given ML problem.

T3 Executable Knowledge: It refers to knowledge elements that can be executed directly or after integration into software systems. These elements often embed both declarative and procedural knowledge, typically in the form of code. Examples include code snippets, ML libraries, and pre-trained ML models. Such elements are reused to reduce implementation effort and accelerate development.

TABLE II. Searched Knowledge Elements.

Category	Knowledge Element Type	#Part.
T1 Declarative Knowledge	ML Algorithm	12
	Foundations	8
	Terminology	7
T2 Procedural Knowledge	Solution Design Pattern	11
	Step-by-Step Instruction	6
T3 Executable Knowledge	Code Snippet or Repository	12
	ML Library	6
	Pre-trained ML Model	4

In practice, ML developers blend different types of knowledge: understanding key concepts (declarative), applying structured approaches or steps (procedural), and adapting existing code (executable).

B. How do ML developers search for knowledge (RQ2)?

To answer this question, the TA team analyzed participants' accounts of their search approaches and grouped the identified codes into three themes (see Table III), which we describe in the following section.

TABLE III. Applied Search Approaches.

Search Approaches	#Part.
T4 Opportunistic Behavior	18
T5 Executable First	10
T6 Solution Idea First	8

T4 Opportunistic Behavior: In line with IFT, the TA team observed *opportunistic search behavior* among all participants in their choice of search and retrieval tools, knowledge sources, and how they evaluated the quality of knowledge elements. Many relied on surface-level quality evaluation, such as recognizing familiar sources or drawing on prior experiences, to rapidly select and evaluate the quality of knowledge elements. This behavior aligns with the concept of information scent, where individuals are drawn to sources that appear to offer valuable and actionable information while requiring minimal cognitive effort.

T5 Executable First: Ten participants mentioned that they first search for executable knowledge that could directly address the problem at hand. This was especially dominant with academic practitioners. For example, P7 noted that understanding why an existing solution works is often easier than defining an entire solution concept from scratch. Two participants, one from industry and one from academia, noted that they did not care about the internal mechanisms as long as the solution effectively addressed the problem. This behavior aligns with opportunistic behavior as these participants favor rapid progress over deeper conceptual understanding.

T6 Solution Idea First: In contrast to T5, eight participants stated that when searching, they first focus on understanding the underlying reasoning behind an existing solution design pattern before translating it into executable code. These approaches were not mutually exclusive: Some participants first explored an executable knowledge element and subsequently worked to understand the conceptual ideas behind it, while

others started with forming a conceptual idea before selecting an executable knowledge element. Both approaches were observed among ML developers from both industry and academia.

C. Where do ML developers search for knowledge, and what tools do they use to do so (RQ3)?

During coding, the TA team identified different knowledge sources as well as search and retrieval tools. These were grouped into the theme *search & retrieval tools*, the purpose-based themes *sources for learning* and *sources for coding*, and the separate theme *human sources*. A complete overview is provided in Tables IV and V.

TABLE IV. TOOLS USED FOR SEARCHING AND RETRIEVING KNOWLEDGE.

T7 Search and Retrieval Tool Type	#Part.
Web Search Engine	18
Scholarly Literature Search Engine	10
LLM	9

TABLE V. USED KNOWLEDGE SOURCES.

Category	Knowledge Source Type	#Part.
T8 Sources for Learning	Blog Article	16
	Scientific Paper	11
	Video	10
	Book	6
T9 Sources for Coding	Stack Overflow Post	14
	Technical Documentation	10
	Personal Code Archive	10
	Git Repository	9
T10 Human Sources	Colleague / Expert	17

T7 Search & Retrieval Tools: ML developers often begin by using web search engines, such as Google, or scholarly literature search engines, like Google Scholar, to investigate unfamiliar problems and identify existing solutions. Nine participants also utilized LLMs (e.g., ChatGPT) for this purpose, highlighting that they enable a more interactive and conversational approach to exploring knowledge elements. Six participants also mentioned using LLMs for code generation and content summarization. Strikingly, a larger proportion of ML developers from academia (4 out of 6) reported using some form of AI assistance compared to their counterparts in industry (2 out of 12).

T8 Sources for Learning: To explore the solution space, ML developers used scientific papers. This tendency was particularly pronounced among academic ML developers (5 out of 6), whereas two participants from industry reported that the knowledge in scientific papers was too specific. Blogs were preferred for clarity and practical examples. Videos were considered useful for hands-on learning and tutorials. However, they were sometimes considered less efficient than skimming text. Books were mentioned as a source of declarative knowledge, but were also said to be more difficult to search through.

T9 Sources for Coding: For coding-related questions, Stack Overflow was the primary source, valued primarily for the community feedback attached to each answer. Technical documentation (e.g., the scikit-learn user guide) was particularly valued for guiding reuse, with eight ML developers from industry and only two from academia. Git repositories offer reusable code, often accessed through links provided in blogs and papers. Ten ML developers reported searching their code archives, although disorganization sometimes limited their usefulness. The executable files within the coding sources are typically "copied and pasted" or downloaded and then modified.

T10 Human Sources: Participants consulted colleagues or experts, although some hesitated to seek help due to concerns about interrupting or disturbing them. Although not discussed in the interviews, human knowledge sources can also share tacit knowledge, i.e., personal experiences with specific solution approaches.

D. How do ML developers evaluate knowledge and its sources to decide whether they want to reuse them (RQ4)?

To answer this research question, the TA team identified codes for each mentioned criterion, by which a knowledge element and its source are evaluated. Then, these criteria were grouped by the overarching concept they describe. As a result, the TA team identified three themes, namely *quality-based*, *intuition-based*, and *experiment-based* evaluation approaches (see Table VI). We describe each approach below.

TABLE VI. APPLIED KNOWLEDGE EVALUATION APPROACHES.

Evaluation Approach	#Part.
T11 Quality-based	18
T12 Experiment-based	15
T13 Intuition-based	10

TABLE VII. QUALITY CRITERIA USED FOR EVALUATING KNOWLEDGE ELEMENTS.

Quality	Quality Criterion	#Part.
T11.1 Usability	Compatibility with development environment	15
	Writing style	14
	Ease of understanding	14
	Presence of examples, code, or tutorials	12
	Effort required to reuse the solution	10
T11.2 Credibility	Recognized author or publisher	14
	Community feedback	12
	Transparency of decisions and results	11
	Match with personal knowledge or experience	8
	Cited or mentioned frequently	6
T11.3 Relevance	Context match	17
	Recency of publication or update	14

T11 Quality-based Evaluation: Three key qualities were used to evaluate knowledge elements and sources: *usability*, *credibility*, and *relevance* (see Table VII).

T11.1 Usability: It refers to how easily an ML developer can understand and apply a knowledge element. This includes, among other things, the perceived ease of understanding, the effort required for reuse, and ease of integration into the ML developer's development environment. If the perceived effort required for reuse is too high, ML developers discard solutions

even when they would yield high performance. This occurs, for example, when programming languages are incompatible.

T11.2 Credibility: It describes the perceived trustworthiness of the source. ML developers consider criteria such as the author's expertise, community feedback, or the number of citations. ML developers tend to favor sources created by recognized experts, which are perceived as more credible due to the assumed rigor of their development, and therefore spend less time critically analyzing them.

T11.3 Relevance: It concerns the degree to which a knowledge element matches with the ML developer's problem context, i.e., problem definition and data. The publication or update date is a frequently mentioned criterion, since older knowledge elements may be outdated. Practitioners typically assess context match first, before investing further effort in analyzing the knowledge element.

T12 Experiment-based Evaluation: In the case of executable knowledge elements, such as code snippets, 15 ML developers prefer to directly experiment with the artifact to evaluate its strengths, limitations, and overall suitability. This also helps them gain a better understanding (P13) or a "better feeling" (P14) of the solution space. This was frequently reported by all ML developers from academia.

T13 Intuition-based Evaluation: ML developers frequently rely on their intuition to make quick judgments about the value of a knowledge element or the credibility of its source. This trend was particularly pronounced among ML developers from industry (7 out of 12), compared to those from academia (3 out of 6). Furthermore, this tendency is more pronounced among ML developers with at least two years of experience (9 out of 10).

E. How do ML developers retain relevant knowledge (RQ5)?

ML developers employ a range of approaches to document and manage relevant knowledge as they solve problems. To develop themes, the TA team grouped codes by the nature of the retention approaches, distinguishing between *absent*, *ad-hoc*, *unstructured*, and *structured* approaches (see Table VIII).

TABLE VIII. APPLIED KNOWLEDGE RETENTION APPROACHES.

Retention Approach	Mentioned Example	#Part.
T14 Absent	Rely on memory only	10
T15 Ad-hoc Bookmarking	Keeping tabs open	7
	Revisiting search history	2
T16 Unstructured Note-taking	Taking digital or physical notes	8
	Leaving comments in code	3
T17 Structured Documentation Tools	Notes in dedicated apps	6
	Curating collection of links	5

T14 Absent: Ten participants relied solely on memory, keeping knowledge elements "in mind" as they worked. Among participants adopting this strategy, seven out of ten were from industry.

T15 Ad-hoc Bookmarking: A commonly reported approach was to rely on the search history or keep browser tabs open, often resulting in what P2 described as "tab chaos". Tabs were used to temporarily store promising knowledge sources, which participants planned to revisit later in the process. These ML developers also reported retaining the information only until the problem was resolved, indicating no intention to reuse the knowledge in the future.

T16 Unstructured Note-Taking: Another approach is to keep physical or digital notes. Eight participants described writing down key ideas and partial solutions in notebooks or using digital note-taking tools, motivated by the desire to explain their design decisions at any time. Three participants also added comments in the code to document the origin or logic of reused code snippets. These notes served both as external memory aids and as a way to organize thoughts during exploration.

T17 Structured Documentation Tools: A subset of participants used more organized approaches for long-term knowledge retention. Commonly mentioned tools included digital note-taking applications with filtering and tagging capabilities, as well as Overleaf, Citavi, and Microsoft Office software.

VI. DISCUSSION

Our sample included participants with diverse backgrounds, roles, and domains. While this variation might be expected to introduce substantial differences in prototyping and reuse practices, our analysis revealed a high degree of consistency across participants. The vast majority of codes occurred in multiple interviews, and the core themes emerged independently of role, experience level, or industry context. This suggests that the identified practices are robust across heterogeneous ML developer groups.

Based on the research questions and study results, we identified the central reuse-related concepts and their relationships (see Figure 1). This conceptual model can be interpreted as follows: Depending on the selected *search approach*, the *knowledge search* is performed using dedicated *search and retrieval tools*. These tools retrieve *knowledge sources* that capture *knowledge elements* of different *knowledge types*. The retrieved knowledge elements are systematically assessed using a selected *evaluation approach* to determine their suitability for the current ML prototyping context. Accepted knowledge elements may be retained using a *retention approach*, thereby facilitating their reuse in subsequent projects or iterations.

A. Implications for Knowledge Providers

By identifying evaluation approaches, we provide insights into how ML developers decide whether to reuse knowledge and which criteria influence these decisions. Knowledge providers can leverage this understanding to strategically strengthen information scent, thereby increasing the likelihood of reuse. Moreover, organizations can use these insights to establish standardized practices for sharing knowledge generated during ML solution prototyping.

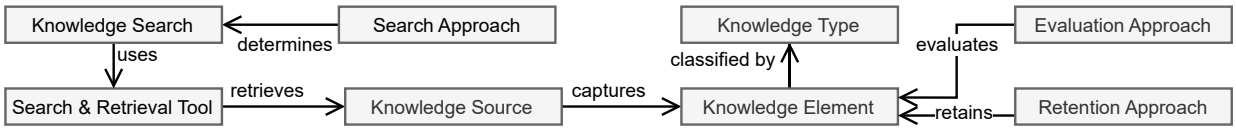

Figure 1. Concept Model of Reuse in ML Solution Prototyping

B. Implications for Tools

More than half of the participants do not employ sustainable methods to retain reusable knowledge. Knowledge thus tends to become siloed at the individual level, either mentally or in personal code archives, as evidenced by the fact that 17 participants reported relying on human sources.

The results of our study highlight several important directions for future tool development to address the challenges of knowledge retention in ML prototyping. The absence of systematic approaches to retaining reusable knowledge is due not only to insufficient tooling but also to limited integration with ML developers' daily workflows and collaborative platforms.

To address these issues, future tools should prioritize features that support *collaborative management* and *long-term preservation* of both knowledge sources and elements. Seamless integration with widely used IDEs and collaborative platforms can help embed retention practices into everyday work. Additionally, *automated tracing* between artifacts and reused knowledge elements and their sources would facilitate better organization and retrieval of information over time. Incorporating mechanisms for *quality-based evaluation* into knowledge retention systems could further support ML developers, particularly given our finding that ML developers rely on numerous quality criteria but lack systematic means to manage them.

By addressing these needs, new tools have the potential to reduce individual knowledge silos, foster broader sharing of valuable knowledge across projects and teams, and ultimately increase the degree of reuse in ML solution prototyping.

VII. Threats to Validity

Internal Validity: To mitigate threats to internal validity, we designed a semi-structured interview aligned with our research questions. Two researchers independently coded all transcripts and resolved differences through discussion to limit individual bias. However, some subjectivity remains inherent in thematic analysis, and our results depend on what participants chose to share during their reflections. Furthermore, internal validity may be affected by recall and social desirability bias, as participants may misremember their actions or present their decisions in a more favorable light. To mitigate this, we assigned participants a prototyping task.

External Validity: Our findings are based on interviews with 18 ML developers from industry and academia, covering a range of domains and roles. While this diversity supports the breadth of our insights, the limited sample size constrains generalizability. Future studies with larger samples could further strengthen the transferability of our findings.

VIII. Conclusion and Future Work

This interview study examines how ML developers search for, evaluate, reuse, and retain knowledge during solution prototyping. By broadening the focus beyond code reuse, our findings reveal that practitioners reuse a broad spectrum of explicit knowledge, including declarative, procedural, and executable knowledge.

Our results show that their search and evaluation behaviors align with the principles of information foraging theory and are strongly influenced by opportunistic decision-making. The entire reuse process is shaped by the trade-off between speed and systematic development. Activities such as understanding the solution, documenting design decisions, and ensuring sustainable knowledge retention are often neglected in favor of quickly producing "good enough" solutions.

Because opportunistic behavior discourages ML developers from investing time in manual knowledge retention, we want to investigate how retention processes can be supported or automated to promote knowledge reuse. Furthermore, we aim to evaluate the validity of our knowledge reuse concept, assessing the extent to which it reflects and accurately represents real-world practices.

References

[1] H. Mili, F. Mili, and A. Mili, "Reusing Software: Issues and Research Directions," *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp. 528–562, 1995. DOI: 10.1109/32.391379.

[2] M. Paul, M. Engelhart, I. Rus, and S. Sinha, "Knowledge Management in Software Engineering - A State-of-the-Art report," Jan. 2001.

[3] A. Dyck, A. Ganser, and H. Lichter, "On Designing Recommenders for Graphical Domain Modeling Environments," in *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development - MODELSWARD*, INSTICC, SciTePress, 2014, pp. 291–299.

[4] L. Heinemann, "Facilitating Reuse in Model-Based Development With Context-Dependent Model Element Recommendations," in *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*, 2012, pp. 16–20.

[5] L. Heinemann, V. Bauer, M. Herrmannsdoerfer, and B. Hummel, "Identifier-Based Context-Dependent API Method Recommendation," in *2012 16th European Conference on Software Maintenance and Reengineering*, 2012, pp. 31–40.

[6] W. Adjandra, Y. Putrapratama, A. Wiraguna, D. Sensuse, and N. Safitri, "Systematic Literature Review Knowledge Reuse in Software Development," in *Proceedings - 2nd International Conference on Computer Science and Engineering*, United States: Institute of Electrical and Electronics Engineers Inc., 2021.

[7] V. Bauer, J. Eckhardt, B. Hauptmann, and M. Klimek, "An Exploratory Study on Reuse at Google," in *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices*, ser. SERIPs 2014, Hyderabad, India: ACM, 2014, pp. 14–23.

[8] V. Bauer and B. Hauptmann, "Assessing Cross-Project Clones for Reuse Optimization," in *2013 7th International Workshop on Software Clones (IWSC)*, 2013, pp. 60–61.

[9] A. Lotter, S. A. Licorish, B. T. R. Savarimuthu, and S. Meldrum, "Code Reuse in Stack Overflow and Popular Open Source Java Projects," in *2018 25th Australasian Software Engineering Conference (ASWEC)*, 2018, pp. 141–150.

[10] S. Mahajan, N. Abolhassani, and M. R. Prasad, "Recommending Stack Overflow Posts for Fixing Runtime Exceptions Using Failure Scenario Matching," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE '20, ACM, Nov. 2020, pp. 1052–1064.

[11] G. Melo, T. Oliveira, P. Alencar, and D. Cowan, "Knowledge Reuse in Software Projects: Retrieving Software Development QA Posts Based on Project Task Similarity," *PLOS ONE*, vol. 15, no. 12, pp. 1–27, Dec. 2020.

[12] C. Ragkhitwetsagul and M. Paixao, "Recommending Code Improvements Based on Stack Overflow Answer Edits," *Computing Research Repository (CoRR)*, vol. 2204, 2022. arXiv: 2204.06773 [cs.SE].

[13] R. Huang *et al.*, "How Scientists Use Jupyter Notebooks: Goals, Quality Attributes, and Opportunities," *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1243–1255, 2025.

[14] A. Koenzen, N. A. Ernst, and M. Storey, "Code Duplication and Reuse in Jupyter Notebooks," *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–9, 2020.

[15] M. S. Siddik, H. Li, and C.-P. Bezemer, "A Systematic Literature Review of Software Engineering Research on Jupyter Notebook," *Computing Research Repository (CoRR)*, vol. 2504, 2025. arXiv: 2504.16180 [cs.IR].

[16] B. van Oort, L. Cruz, M. Aniche, and A. van Deursen, "The Prevalence of Code Smells in Machine Learning projects," in *1st IEEE/ACM Workshop on AI Engineering - Software Engineering for AI, WAIN@ICSE 2021, Madrid, Spain, May 30-31, 2021*, IEEE, 2021, pp. 35–42.

[17] M. Yang, Y. Zhou, B. Li, and Y. Tang, "On Code Reuse from StackOverflow: An Exploratory Study on Jupyter Notebook," *Computing Research Repository (CoRR)*, vol. 2302, 2023. arXiv: 2302.11732 [cs.SE].

[18] M. Källén, U. Sigvardsson, and T. Wrigstad, "Jupyter Notebooks on GitHub: Characteristics and Code Clones," *Computing Research Repository (CoRR)*, vol. 2007, 2020. arXiv: 2007.10146 [cs.SE].

[19] W. Epperson, A. Y. Wang, R. DeLine, and S. M. Drucker, "Strategies for Reuse and Sharing among Data Scientists in Software Teams," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '22, Pittsburgh, Pennsylvania: ACM, 2022, pp. 243–252.

[20] X. Li, Y. Wang, H. Wang, Y. Wang, and J. Zhao, "NBSearch: Semantic Search and Visual Exploration of Computational Notebooks," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI '21, ACM, May 2021, pp. 1–14.

[21] M. Horiuchi, Y. Sasaki, C. Xiao, and M. Onizuka, "JupySim: Jupyter Notebook Similarity Search System.," in *EDBT*, 2022, pp. 2–554.

[22] Elyra Team, *Code Snippets - Elyra 3.15.0 documentation*, https://elyra.readthedocs.io/en/v3.15.0/user_guide/code-snippets.html, [Acc. 03-Apr-2026], 2022.

[23] C. Ragkhitwetsagul *et al.*, "Typhon: Automatic Recommendation of Relevant Code Cells in Jupyter Notebooks," *Computing Research Repository (CoRR)*, vol. 2405, 2024. arXiv: 2405.09075 [cs.SE].

[24] A. Watson, S. Bateman, and S. Ray, "PySnippet: Accelerating Exploratory Data Analysis in Jupyter Notebook through Facilitated Access to Example Code," in *EDBT/ICDT Workshops*, 2019.

[25] X. Li, Y. Zhang, J. Leung, C. Sun, and J. Zhao, "EDAssistant: Supporting Exploratory Data Analysis in Computational Notebooks with In Situ Code Search and Recommendation," *ACM Trans. Interact. Intell. Syst.*, vol. 13, no. 1, Mar. 2023, ISSN: 2160-6455.

[26] L. Li and J. Lv, "Unlocking Insights: Semantic Search in Jupyter Notebooks," *Computing Research Repository (CoRR)*, vol. 2402, 2024. arXiv: 2402.13234 [cs.SE].

[27] E. Peixoto, D. Torres, D. Carneiro, B. Silva, and R. Marques, "Reusing ML Models in Dynamic Data Environments: Data Similarity-Based Approach for Efficient MLOps," *Big Data and Cognitive Computing*, vol. 9, no. 2, 2025, ISSN: 2504-2289.

[28] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

[29] R. Ahmad, N. N. Manne, and T. Malik, "Reproducible Notebook Containers using Application Virtualization," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 1–10.

[30] A. Rule *et al.*, "Ten Simple Rules for Reproducible Research in Jupyter Notebooks," *Computing Research Repository (CoRR)*, vol. 1810, 2018. arXiv: 1810.08055 [cs.OH].

[31] J. Wang, T.-y. Kuo, L. Li, and A. Zeller, "Assessing and Restoring Reproducibility of Jupyter Notebooks," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '20, Virtual Event, Australia: ACM, 2021, pp. 138–149.

[32] P. Pirolli and S. Card, "Information Foraging.," *Psychological Review*, vol. 106, no. 4, p. 643, 1999.

[33] O. Brown, N. Power, and J. Gore, "Cognitive task analysis: Eliciting expert cognition in context," *Organizational Research Methods*, vol. 28, no. 3, pp. 375–404, 2025.

[34] V. Braun and V. Clarke, *Successful Qualitative Research: A Practical Guide for Beginners*. Sage Publications ltd, 2013.

[35] Delve, *Qualitative Data Analysis Software Delve*, https://delvetool.com, [Acc. 03-Apr-2026], 2025.

[36] S. Coban and P. Chrestin, *Artifacts from the Interview Study: Knowledge Reuse in ML Solution Prototyping*, Zenodo, Jan. 2026. DOI: 10.5281/zenodo.18196720. [Online]. Available: https://doi.org/10.5281/zenodo.18196720.

[37] T. Ten Berge and R. Van Hezewijk, "Procedural and Declarative Knowledge: An Evolutionary Perspective," *Theory & Psychology*, vol. 9, no. 5, pp. 605–624, 1999.