information

MDPI

Article

Business Logic Vulnerabilities in the Digital Era: A Detection Framework Using Artificial Intelligence

Bilgin Metin [1], Martin Wynn [2,*], Aylin Tunalı [1] and Yağmur Kepir [1]

[1] Department of Management Information Systems, Bogazici University, Istanbul 34342, Turkey; bilgin.metin@bogazici.edu.tr (B.M.); yagmur.kepir99@gmail.com (Y.K.)
[2] The School of Business, Computing and Social Sciences, University of Gloucestershire, Cheltenham GL50 2RH, UK
* Correspondence: mwynn@glos.ac.uk

Abstract

Digitalisation can positively impact the efficiency of real-world business processes, but may also introduce new cybersecurity challenges. One area that is particularly vulnerable to cyber-attacks is the business logic embedded in processes in which flaws may exist. This is especially the case when these processes are within web-based applications and services, which are increasingly becoming the norm for many organisations. Business logic vulnerabilities (BLVs) can emerge following the software development process, which may be difficult to detect by vulnerability detection tools. Through a systematic literature review and interviews with industry practitioners, this study identifies key BLV types and the challenges in detecting them. The paper proposes an eight-stage operational framework that leverages Artificial Intelligence (AI) for enhanced BLV detection and mitigation. The research findings contribute to the rapidly evolving theory and practice in this field of study, highlighting the current reliance on manual detection, the contextual nature of BLVs, and the need for a hybrid, multi-layered approach integrating human expertise with AI tools. The study concludes by emphasizing AI's potential to transform cybersecurity from a reactive to a proactive defense against evolving vulnerabilities and threats.

Keywords: business logic vulnerabilities; BLVs; business processes; organisational cyber security; artificial intelligence; AI; operational framework

Academic Editor: Kostas Vergidis

Received: 31 May 2025
Revised: 1 July 2025
Accepted: 3 July 2025
Published: 7 July 2025

Citation: Metin, B.; Wynn, M.; Tunalı, A.; Kepir, Y. Business Logic Vulnerabilities in the Digital Era: A Detection Framework Using Artificial Intelligence. *Information* **2025**, *16*, 585. https://doi.org/10.3390/info16070585

1. Introduction

As digitalisation expands across business and society at large, more services and products are being delivered to customers via the Internet through mobile and web-based platforms. This growth in web-based transactions has seen a rapid increase in cyber-attacks on web applications. SiteLock [1] reported that there were an estimated 12.8 million websites that had been compromised with malware, and many key business processes now involve the use of web-based applications in the cloud, representing potential points of weakness that cyber-attackers may exploit. This includes business logic vulnerabilities (BLVs)—flaws in the design, implementation, or enforcement of business rules within software applications, which can be exploited to perform unintended operations within the software [2]. As the complexity of business processes executed by web applications has increased, so has the risk of potential flaws in the process logic [3]. Business logic attacks are among the top ten most frequent vulnerabilities, accounting for 3% of all vulnerabilities reported through the HackerOne platform [4].

BLVs pose a significant threat to modern web applications and enterprise systems. Unlike traditional security vulnerabilities, business logic flaws are normally not inherently linked to coding errors or software bugs, but rather often result from incorrect or incomplete implementation of business rules and workflows. BLVs may allow attackers to change an application's business logic that may be exploited via seemingly valid application transactions that can be utilised to accomplish an additional action, outside the scope of regular business operations. As Hofesh [5] notes "unlike most other vulnerabilities, business logic exploits are heavily contextualized and very difficult to detect automatically, leaving room for different interpretations of the application's logic, which could go unnoticed by the developers" (paragraph 1).

Manual techniques and procedures have hitherto often been deployed for detecting BLVs. The Open Web Application Security Project (OWASP) previously concluded that automating the identification and resolution of business logic abuse cases is not feasible. This is because such measures rely heavily on the expertise of the tester, as well as their comprehensive understanding of the entire business process and its rules [6]. Furthermore, identifying and mitigating BVLs often depends on the manual expertise of testers who possess comprehensive knowledge of the application's business logic and the processes they support [7,8]. Manual testing expertise can be assessed and accredited via penetration testing certifications. To become a certified penetration test expert, there are several well-recognized certifications, each with their own set of requirements and focus areas [9]. Two of the most prominent certifications are the Offensive Security Certified Professional (OSCP) and the Certified Ethical Hacker (CEH) qualifications, which are highly regarded and often required for penetration testing roles in industry [10]. The related examinations involve hands-on practice, and only proven experts are typically capable of conducting thorough penetration tests.

The impact of such vulnerabilities can be severe, potentially allowing attackers to manipulate legitimate functionalities, execute unauthorised transactions, and disrupt business operations [2,6,11]. However, OWASP have more recently put forward a Web Security Testing Guide (WSTG) which offers both manual and automated methodologies for testing business logic, emphasising the importance of addressing unexpected user behavior, privilege escalation, and process exploitation [12]. An example of combining both manual and automated testing methodologies is the Selenium tool version 4.0 [13], which simulates real user behavior, such as clicking, filling out forms, and submitting invalid data. This helps identify failures in handling edge cases or malicious inputs. Data-driven techniques rely on collecting information about software, such as program invariants, to identify BLVs. Fortunately, these logic vulnerabilities can be detected by analyzing the program invariants dynamically using appropriate tools, of which Daikon is one example [14]. Pei et al. [15] also show that Large Language Models (LLMs) can also be used to analyze program invariants.

Artificial Intelligence (AI) is now recognized as a key component of technology support for business operations [16]. AI has been applied to improve business operations, notably in the context of fraud detection, and financial distress prediction [17,18]. The strategic importance of AI for small and medium-sized enterprises (SMEs) is underscored by research focusing on how to accelerate their AI-based product development and innovation processes, particularly in competitive sectors like Fin-Tech [19]. The high-speed, efficiency-focused nature of development and management that is prevalent in many SMEs often results in the absence of comprehensive internal control systems, leading to limited risk awareness among managers and employees [18]. In such an environment, an AI-based vulnerability automation system is particularly beneficial in that it can systematically detect threats and enforce security logic without reliance on personal risk awareness. AI-driven tools have been deployed to strengthen protection against cyber threats and reduce the

risk of data security breaches [20]. However, to date, their application in the context of BLVs has been minimal, and yet it is becoming increasingly clear that AI can now be used to learn and detect logic patterns, making it possible to automate and accelerate process logic testing that has hitherto been carried out manually. Indeed, Metin et al. [21], in their study of cybersecurity in SMEs, note that "whilst there is a substantial amount of research in the literature that looks into different aspects of cybersecurity related issues and practices, the number of studies that approach the topic from the perspective of processes remains limited" (p. 4). In this context, this research explores how the implementation of AI applications can detect and prevent business logic flaws in company business processes and thereby improve cybersecurity structures. More specifically, the following research questions (RQs) are addressed:

RQ1. What are the main types of business logic vulnerability and what are the detection challenges?

RQ2. What operational framework can be developed to guide Al-supported business logic vulnerability detection?

In summary, the detection of BLVs has traditionally been a manual and resource-intensive process, as existing automated tools often fail to identify these complex and subtle flaws because of their specific business context. As a result, there is a significant void because of the recognised potential of AI in cybersecurity and yet the absence of a structured, operational roadmap for its specific application to BLV detection from a business process perspective. The overall objective of this research is thus to address this gap in the literature and practice by analysing BLVs and then to construct an operational framework to guide the systematic implementation of AI-supported BLV detection. The framework can provide a valuable link between theory and practice, synthesizing findings from academic literature with the real-world experiences of industry practitioners. This approach offers organisations an actionable, multi-layered strategy. Ultimately, this contribution enables organisations to shift from reactive detection methods to a more proactive and resilient security posture against business logic threats.

Following this brief introduction, Section 2 sets out the research methodology, centering on a systematic literature review (SLR) complemented by practitioner interview evidence. In Section 3, the study results are set out, and the research questions are addressed. Section 4 then discusses some tangential issues emerging from the research results. In the concluding Section 5, the contribution of the research is assessed, limitations are outlined, and future research areas are noted.

2. Research Method

An explanatory-sequential mixed-methods design was adopted, in which an SLR [22] was combined with a set of practitioner interviews to validate, challenge, and elaborate upon the SLR findings. The SLR was the central component of the research method, but practitioner interviews were used to provide additional material in addressing the two RQs and particularly in validating and evolving the operational framework for BLV detection. This research process is depicted in Figure 1.

2.1. Systematic Literature Review

For the SLR, several research databases were accessed, including IEEE, Science Direct and SCOPUS. The search items used are shown in Table A1 (Appendix A), these being combinations of "Business Process Attacks", "Artificial Intelligence", "Generative Artificial Intelligence" and "Business Logic Vulnerabilities". The total number of sources located was 1152. Inclusion and exclusion criteria are shown in Table 1.

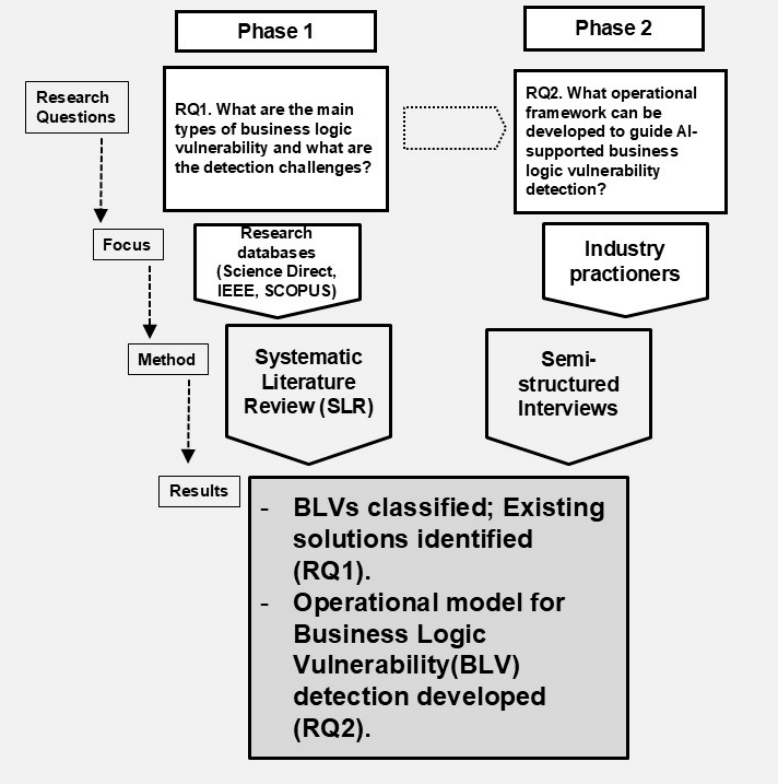

Figure 1. Research process overview.

Table 1. Inclusion and exclusion criteria.

Included	Excluded
Papers published in English	Papers not published in English
Studies focusing on business logic vulnerabilities in organizations and AI implementation in order to reduce cyber risks	Studies only focusing on the negative impact of AI to organizational security
Contents that are categorized in conferences, journal articles, and books from databases such as IEEE, Scopus, and ScienceDirect	Studies not focusing on business logic vulnerabilities in organizations and AI implementation in order to reduce cyber risks
Literature available through open access or university library access through research databases	Sources that cannot be accessed open access or through using our university library
Literature sources published between 2010 and 2024	Literature published before 2010

The PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) Flow Diagram summarises the overall SLR process, which comprises four main phases—identification, screening, eligibility, and synthesis (Figure 2). Following the removal of duplicate entries, the count was reduced to 410 potential sources. Subsequently, 152 records were discarded after reviewing abstracts and keywords, and after eliminating those with restricted access, a total of 37 articles remained.

Of the 37 full-text sources included in the final review, 16 were published between 2016 and 2020, and 10 in the period 2020–2024. This probably reflects the growing interest and academic productivity in AI and cybersecurity in the past decade. 19 of the 37 sources were conference papers, 14 were published journal articles, and the remainder were from various sources (a book, a thesis, a preprint article, and a newsletter article).

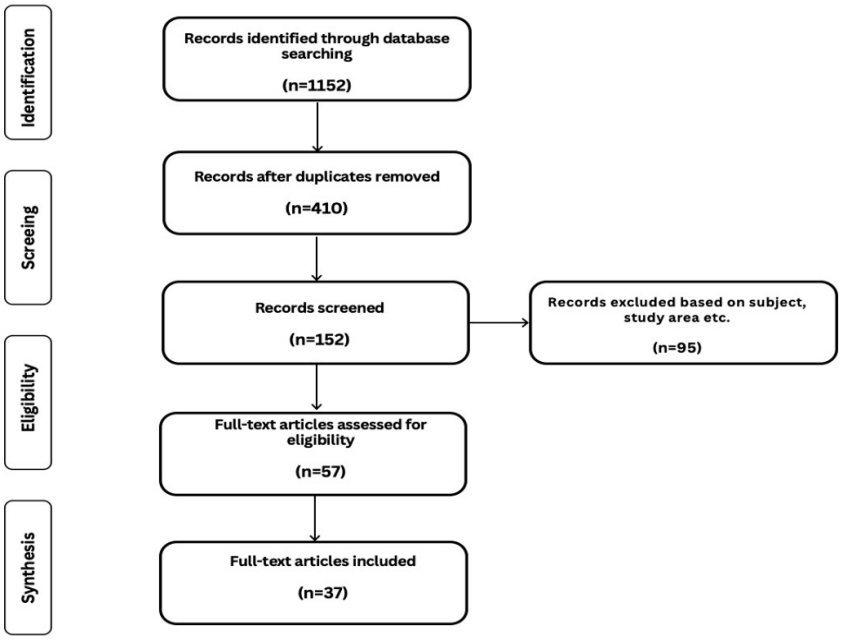

Figure 2. The PRISMA flow diagram for the Systematic Literature Review.

2.2. Qualitative Interviews

Following the SLR, which highlighted several key issues about how BLVs are detected and mitigated in real-world settings, semi-structured interviews were conducted to obtain practitioner perspectives. The interviewees were selected as representatives of different industry sectors and size of company, varying from start-ups to well-established companies with large development teams (Table 2). The interviewees' experience in their IT environment was between 3 and 27 years. Ideally, more interviews would have been conducted, but time constraints and other practicalities, such as the availability and accessibility of qualified participants, the sensitivity of the subject matter (i.e., business logic vulnerabilities), and scheduling difficulties across different organisations, limited the interview count to seven. Although there were some limitations, participants were carefully selected to ensure diversity in organisational roles, sectors, and company sizes, which enriched the insights gathered. This diversity served to contextualise, triangulate, and extend the insights derived from the SLR. The participants included developers, team leads, and quality assurance specialists.

Each interview lasted approximately 40 min and was conducted remotely via video conferencing. Informed consent was obtained prior to recording, and all sessions were transcribed verbatim for analysis. The interviews followed a semi-structured guide consisting of open-ended questions grouped into three thematic areas: detection practices and challenges related to BLVs, common types and impacts of these vulnerabilities, and strategies for their mitigation.

Thematic analysis was used to analyze the data, following the six-phase process outlined by Braun and Clarke [23]. This process began with familiarization with the dataset, during which transcripts were read multiple times to gain an overall understanding of the content. This was followed by initial coding, where meaningful segments were labeled with descriptive codes. These codes were then organized into broader patterns, resulting in a number of primary themes and sub-themes being identified. The research team took the view that seven in-depth interviews with practitioners from different size companies in various sectors were enough to allow the development of new material to address the research questions. This is supported by Guest et al. [24], who found that "basic elements for meta themes were present as early as six interviews" (p. 59). Given the breadth of

feedback obtained through the interviews, the research team considered that "theoretical saturation", which Flick [25] (p. 429) defines as "the judgment that there is no need to collect further data", had been achieved.

Table 2. Demographic and professional profile of interview participants.

Code	Experience (yrs)	Age	Gender	Role/Position	Team Size/ Context
P1	3	26	Female	Co-Founder & Chief Developer	3-person team
P2	3	37	Female	Co-Founder & Chief Developer	3-person core start-up
P3	17	45	Male	Senior Development Team Leader	5-person team
P4	13	51	Male	Software Development Team Leader	28-person team
P5	10	38	Male	Software Development Team Leader/Partner	15-person team
P6	27	45	Male	Software Development Team Manager & Company Owner	6-person team
P7	23	47	Male	R&D & Software-Quality Development Team Manager	6-person team

Throughout the analysis, special attention was paid to ensuring that the identified themes accurately reflected the participants' lived experiences rather than being influenced by the researchers' assumptions. Summaries of the key themes and corresponding quotes were shared with the interviewees, and their feedback was integrated into the final version of the thematic structure. This primary research phase not only validated and expanded upon the findings of the SLR but also provided rich, contextual insights that directly informed the design and validation of the operational framework for BLV detection.

3. Results

3.1. Introduction

The available literature revealed a range of definitions and statements regarding the meaning and context of BLVs. Pellegrino and Balsarotti [26] point out that there is no universally agreed-upon definition of "business logic vulnerabilities" in the literature, and that it is often described in terms of the type of attack it entails. PortSwigger [27] suggests that "business logic" refers to the set of rules that define how an application operates. As these rules are not always directly related to a specific business function or process, the associated vulnerabilities are also known as "application logic vulnerabilities" or simply "logic flaws". Logic flaws are often not evident to users or support personnel as they typically will not be exposed by normal use of the application. However, an attacker may be able to exploit these logic flaws by interacting with the application in ways that developers had not intended. Indeed, several authors highlight the fact that such vulnerabilities can be exploited by attackers to perform unintended operations that deviate from normal business practices [6,7,11,28]. Some of the key citations from these sources are included in Table A2 in Appendix A. The RQs are now addressed in turn in the following two sub-sections, drawing upon both findings from the literature review and interview analysis.

3.2. RQ1. What Are the Main Types of Business Logic Vulnerability and What Are the Detection Challenges?

BLVs represent a significant challenge for organisations, requiring a comprehensive approach to detection and prevention. An understanding of the various types of attacks and implementation of appropriate prevention systems can help protect a company's applications and business processes from potential exploitation. From analysing a range of sources, five main categories of BLVs can be identified: workflow/application flow bypass; race condition issues; business logic-based denial-of-service; parameter manipulation; and access control bypass. Deepa and Thilagam [2] identified three main types of BLVs, namely workflow bypass or misuse; parameter manipulation (also sometimes called tampering attacks); and access-control violations (with authentication and authorization bypass). Ghorbansadeh and Shahriari [11] similarly recognized these three types of logic vulnerability. Alidoosti et al. [29] reference the excessive traffic that can be created on websites due to poorly defined business rules, which means that BLVs might lead to denial-of-service (DoS). For example, if there is no limit to the number of comments received concerning a product on an online shopping site within one second, malicious attackers can exploit this vulnerability to create excessive traffic on the site. Alidoosti et al. [30] also identified the situation in which multiple threads or processes access shared resources concurrently without proper synchronisation. This vulnerability, termed a "race condition issue", may lead to unpredictable and erroneous behaviour in the business logic layer of the application [31]. To prevent this from happening, simultaneous requests and their priority should be clarified within the framework of business rules and the structure should be built accordingly. For example, if multiple users attempt to apply a discount code simultaneously, without proper synchronisation, this could lead to the discount being applied multiple times or to the wrong user, resulting in financial loss for the business.

Research publications on BVLs suggest a range of possible preventative measures, from the development of innovative detection frameworks and tools to detailed analyses of specific threats. Identifying logic flaws requires knowledge of both the intended operation of the application and the actual behavior as implemented in the application [32]. Four main approaches can be discerned. Security-knowledge databases can provide a structured framework for detecting and mitigating vulnerabilities in business logic, allowing companies to address potential flaws proactively [33]. Dynamic security testing techniques, such as the Business-Layer Session Pussling Racer [31], offer dynamic testing against session race conditions. This ensures that business processes remain robust and secure against timing-based manipulations. In addition, black-box detection approaches can also play a crucial role in identifying logic vulnerabilities in web applications. Tools like DetLogic allow for detecting these vulnerabilities without requiring access to the source code, making it easier to identify and address potential issues [34]. Automated systems, such as LogicScope, enhance this process by automating the discovery of logic vulnerabilities, thereby reducing the time and effort needed to identify and mitigate risks [32]. Finally, integrating self-protection mechanisms within software systems can significantly enhance security. These mechanisms enable real-time detection and response to business logic attacks, providing an additional layer of protection [7]. To combat business-layer DoS attacks, specific mitigation measures can be implemented to increase the resilience of web applications. These measures involve monitoring and limiting the impact of suspicious activities to maintain service availability [29].

There are a number of related concepts in the extant literature. Taubenberger et al. [35] refer to Business Process Vulnerability (BPV), which the authors see as wrongly identified or unidentified vulnerabilities in information security risk assessments. These, the authors suggest, can be resolved by analysing the security requirements of information assets in business process models. Merrell and Stevens [36] discuss a vulnerability management process, which comprises four main stages: the preparation stage, which includes creating

and implementing a vulnerability management plan; the identification and analysis phase, which examines infrastructure vulnerabilities and evaluates whether a particular vulnerability needs more attention; the exposure management phase then ensures that vulnerability repair efforts are based on organisational risk; and finally, the root-cause analysis phase aims to understand how vulnerabilities are introduced into the environment and how to prevent them in the future. In a wider context still, Al-Turkistani et al. [37] examine organisational cybersecurity management and suggest organisations must adopt a range of policies and practices. These include: adherence to predefined procedures to maintain frequent system upgrades and enhancements; evaluation of current security controls; compliance with established security standards; creation of integrated risk assessments and implementation of unified controls; and establishment of incident response management.

The inadequacies of cybersecurity vulnerability scanner systems currently deployed highlight the potential benefits of the integration of AI into such systems. Vulnerability scanners like Nessus, OpenVAS, and Nmap Scripting Engine (NSE) often rely on other third-party "plug-in" software to detect issues such as software bugs, missing OS patches, insecure configurations, and exposed ports or services. In large networks, these tools can generate an overwhelming volume of such data, requiring substantial human interpretation to distinguish true vulnerabilities from false positives, often caused by incomplete or missing plug-ins [38]. Furthermore, they cannot predict or adapt to new and evolving threats, limiting their effectiveness in dynamic environments. These tools also lack the capability to automate comprehensive risk assessments, requiring manual aggregation and interpretation of scan results to determine overall security posture [39]. These related concepts and practices provide a useful background and context to address the specific issue of BLVs and how they can best be protected against.

In this context, the interviewees highlighted the complexity of BLVs and identified a number of detection challenges (Table 3). All seven interviewees framed BLV discovery as an essentially human-driven activity that competes with chronic time and resource constraints. Senior managers in larger teams described an environment in which "we rely exclusively on manual assessments to detect BLVs; automated tools create too many false positives" (P7 in Table 2), and where under-funded start-ups often "discover BLVs in production" because a dedicated QA function is a "luxury" (P5). Smaller firms echoed the same tensions, attributing missed flaws to incomplete or poorly communicated requirements (P6) and developer blindness—the tendency to overlook logic they personally authored (P7). Automation, while present in CI/CD pipelines, was said to have "blind spots for business logic" (P2), because tools verify code syntax rather than usage context. Collectively, these accounts depict BLV detection as an artisanal craft overshadowed by delivery pressure, fragile documentation, and the limits of current scanning technology.

Interviewees also highlighted how complexity compounded the detection process. When asked which flaws are hardest to manage or detect, practitioners converged on a small set of high-impact but low-visibility categories. Access-control lapses topped the list—e.g., an SaaS client who "was able to view data they were not authorised to access" (P4), because of a single permission. Equally pernicious are time-sensitive financial rules: missing a quarterly price update rendered an entire quoting engine inaccurate (P7). Several interviewees noted concurrency exploits, such as race conditions that allow a balance to be spent multiple times (P6), and fraud logic failures where stolen credit cards slip through undetected (P3), as particular problems. In addition, "silent" integration limits—like an undocumented 10-megabyte application program interface (API) cap that truncates responses without errors (P3)—illustrate how BLVs can remain invisible until manifested as business losses. What unites these cases is not sophisticated code manipulation but subtle mismatches between business rules, system timing and real-world behaviour—the very space where traditional vulnerability scanners offer little coverage.

Table 3. Themes identified from interviews relating to BLVs and detection challenges (RQ1).

Primary Theme	Sub-Theme	Relevant Interviewee Quotation	Interpretation
Detection Challenges	Manual-first detection culture	"We rely exclusively on manual assessments to detect BLVs . . . automated tools create too many false positives." P7	Reliance on human judgement introduces scalability, cost and consistency issues.
	Limited QA resources & time pressure	"In start-ups . . . many BLVs are discovered in production because we can't cover every edge path." P5	Budget and schedule pressure shift discovery to post-release.
	Incomplete or misunderstood requirements	"BLVs often result from human error, such as incomplete or incorrectly communicated requirements." P6	Early analysis flaws propagate into hidden logic issues.
	Developer blindness/familiarity bias	"Developers may be unable to recognise flaws in the logic they themselves implemented." P7	Insider perspective masks hidden assumptions.
	Automation blind spots	"Automated testing and scanning tools are typically ineffective . . . the system behaves correctly under standard conditions." P2	Current scanners lack business-context understanding, forcing manual fallback.
Nature and Complexity of BLVs	Access-control & authorisation bypass	"A customer was able to view data they were not authorised to access due to a permission misconfiguration." P4	Mis-scoped privileges silently expose data.
	Time-sensitive financial logic gaps	"Quarter-based pricing logic was missed, so the system failed to reflect time-sensitive costs." P7	Overlooked temporal rules create direct revenue impact.
	Concurrency/race-condition exploits	"A race condition let the same balance be used multiple times, creating funds that do not actually exist." P6	State timing issues allow monetary abuse.
	Undetected fraud scenarios	"Failure to detect fraudulent credit-card use can lead to significant financial loss." P3	Fraud logic needs behavioural context as well as code checks.
	Hidden integration limits	"An undocumented 10 MB API limit silently truncated data, making the issue hard to detect." P3	Third-party constraints become latent BLVs when unvalidated.

3.3. RQ2. What Operational Framework Can Be Developed to Guide AI-Supported Business Logic Vulnerability Detection?

In a general sense, it is clear that business continuity planning can benefit from the incorporation of AI-powered cybersecurity to improve an organisation's capacity to identify and address cybersecurity issues. With such integration, cybersecurity measures can be both preventive and reactive, addressing threats before they have a substantial negative impact [40]. BLVs represent a critical and often overlooked category of security flaws that do not usually arise from syntactic coding errors but rather from misaligned or poorly enforced business rules. Unlike traditional software vulnerabilities, BLVs are deeply contextual and frequently mimic legitimate user behavior, making them difficult to detect using conventional static and dynamic analysis tools. To address this challenge, an AI-supported BLV detection framework is put forward here, based on an analysis of best practice documented in current sources and supported by practitioner interview evidence.

The framework comprises eight stages, for which various technologies may be applied in different process areas (Figure 3).

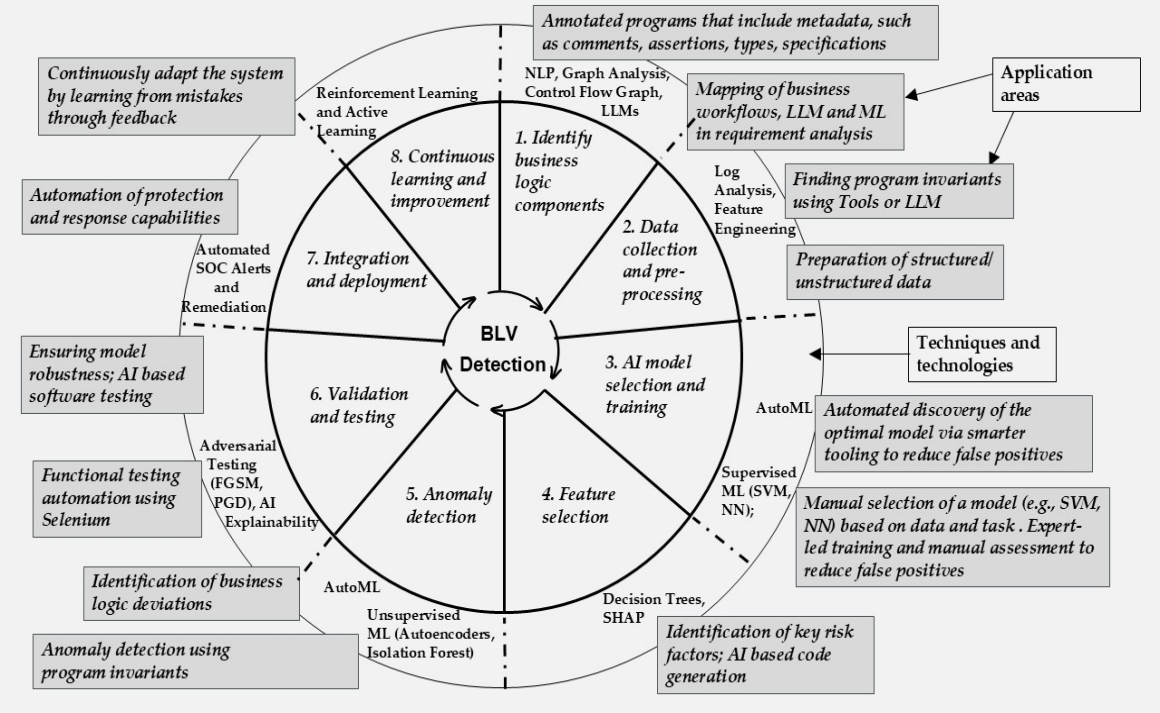

Figure 3. AI-supported BLV detection framework (indicating the 8 stages, with technology options and application areas).

1. Identify business logic components: A comprehensive and detailed mapping of all business processes is required, highlighting how different activities, such as user authentication and data validation, interact with each other. This approach helps in understanding the overall structure and flow of operations and determining potential points of failure [41]. Ghorbansadeh and Shahriari [11] present a method for analyzing source code to detect logic vulnerabilities through data and control flow analysis. This approach is applied to annotated programs, which include additional information or metadata—such as comments, assertions, types, specifications, or other annotations—that assist in guiding the analysis, verification, or transformation of the code. Interviewees confirmed that such activities were being pursued often in the context of software development. P7, for example, noted "business flows are modelled in UML activity/sequence diagrams".

2. Data collection and pre-processing: For each business process, historical data, including logs, user interactions, and transaction records from various sources, such as server logs, application logs, and database transactions, need to be collected. This was emphasised by P2, who commented that "insufficient backend logging in Firebase makes BLVs hard to detect". Also, P5 observed that "most logic flaws tend to surface during functional testing. We use the Selenium webdriver tool [13] to simulate real user behavior—like clicking, form-filling, or submitting invalid data—to detect edge-case or malicious input handling failures". This aligns with Felmetsger et al. [14], who noted that exercising the application in a "normal" way to identify logic flaws cannot be fully automated and often requires human assistance—though tools like Selenium can facilitate scripting realistic user interactions. Indeed, Felmetsger et al. [14] introduced an automated method for detecting logic vulnerabilities, starting with dynamic analysis. Via this approach, Chicory [42], Daikon's Java front-end application code, is modified to collect execution traces that provide details about variable values during regular operation. Daikon then processes these runtime traces to infer low-level behavioral patterns, or "likely

program invariants", which serve as a behavioral fingerprint of normal program execution [14]. Execution-trace processing therefore consists of (i) instrumenting each method so that, at run time, it emits the values of all parameters, return values, and relevant object fields at both entry and exit points, and (ii) serialising these snapshots into a trace file that records the chronological sequence of state changes observed during typical user interactions. As Felmetsger et al. [14] (p. 5) remark, "Chicory produces traces only for procedure entry and exit points and non-local variables", enabling Daikon to generate invariants for "method parameters, function return values, static and instance fields of Java objects, and global variables". Internally, Daikon "runs a program, observes the values that the program computes, and then reports properties that were true over the observed executions" [42] (p. 1). The result is "a file containing a serialized version of likely invariants for the given web application" [14] (p. 6), effectively a machine-checkable behavioural specification that can provide a baseline for later model-based vulnerability detection. These Daikon-inferred invariants are treated as implicit behavioral specifications, establishing a behavioral baseline. Subsequently, their tool, Waler, employs model checking to identify program paths that violate these specifications, thereby flagging potential logic vulnerabilities [14]. Daikon can thus play a role in the Data Collection and Preprocessing phase, as it processes execution traces to infer likely invariants that capture the application's observed behavior, providing input for subsequent model-based logic vulnerability detection.

3. AI model selection and training: Appropriate AI models can be trained using the collected data. For example, records of all login attempts, successful and failed transactions, and error logs can be collected and used to train an AI model [43]. Known vulnerabilities, such as unauthorised access or fraudulent transactions, can also be used to train the AI models [44]. Appropriate data mining techniques such as neural networks, decision trees, or support vector machines can be chosen based on the nature of the data and the specific requirements of the vulnerability detection task. The selected models should be trained using labeled data with previously identified vulnerabilities [45]. Ensuring diversity in the training data to cover a wide range of potential vulnerabilities is particularly important. This was emphasized by P7 who stated that "automated tools create too many false positives" and thus "we rely exclusively on manual assessments". Overall, interviewees confirmed the need for smarter, context-aware tooling. At this point, Automated machine-learning (AutoML) frameworks—such as open-source, distributed, in-memory H2O platform—automatically handle data preparation, algorithm selection, and hyperparameter tuning. Using an AutoML framework at Stage 3 (AI model selection and training) enables practitioners to train and rapidly deploy high-performing predictive models with minimal manual effort, reduce false positives, and eliminate manual feature selection [46,47].

As noted above in stage 2, identifying invariants is a key task in program analysis, with applications such as Daikon being used for bug detection, vulnerability assessment, and formal verification. This process relies on dynamic analysis, which requires collecting execution traces from multiple programs runs to generate reliable invariants. Pei et al. [15] explored the use of large language models for program invariant prediction. Their study found that models trained on source code and fine-tuned for invariant generation can successfully perform invariant prediction in a manner akin to static analysis, rather than relying solely on dynamic analysis.

4. Feature selection: Relevant features from the data indicative of business logic flaws should be extracted. This would include, for example, measurable attributes that embody the rules of a mission-critical workflow and whose deviation reliably signals a business-logic flaw. Prior studies [2] show that parameters enforcing business-related limits (for instance, price, quantity or age ranges) and "state variables", such as role and user-id, govern access decisions, so tampering with them directly undermines the intended behaviour of an application. Empirical evidence [34] further demonstrates that out-of-range values (e.g., age = -5), missing validation of the $\langle$role, user-id$\rangle$ session pair, or bypassing CSRF

(Cross Site Request Forgery) token checks expose parameter-manipulation, access-control and workflow-bypass defects. In the context of training AI models that automatically detect such flaws, Saabel et al. [48] advise that "the first important step for our methodology is selecting the most relevant features to minimize training loss and to make AI interpretations easier" (p. 17). Consequently, only attributes tightly coupled to business invariants such as login-attempt frequency, value-range checks, session-role consistency, and sanctioned page-transition sequences, should be promoted to the feature set, ensuring that the AI focuses on anomalies that truly matter for business-logic vulnerability detection. Domain knowledge can be used to select features that highlight anomalies in business operations, and P7 emphasised the importance of "identifying mission-critical processes" for examination for business logic flaws.

5. Anomaly detection: Appropriate techniques can be employed to identify deviations from normal business logic patterns. Statistical methods and machine learning algorithms can be used to detect when the system behaves in unexpected ways. For example, unusually high transaction amounts that deviate from a user's normal spending pattern can be detected, and P3 also reported that credit-card fraud cases often go unnoticed. Unsupervised learning methods can be used to detect previously unknown vulnerabilities [49]. Here, Daikon [14,42] can also be used, as the likely invariants it generates from normal execution traces serve as implicit behavioral baselines; violations of these invariants during later executions represent deviations from expected logic, analogous to anomalies. Here again, AutoML frameworks create automated algorithms to improve performance, reduce false positives, and eliminate manual feature selection [46,47].

6. Validation and testing: Ensuring the model's accuracy by testing it on data that was not used during training helps in evaluating its real-world applicability. Detailed testing should be performed to ensure the model accurately identifies vulnerabilities without false positives [50]. P5 observed that layered testing was essential to confirm that the model correctly identifies true vulnerabilities while minimising false alarms that could cause unnecessary alerts, noting "unit tests prevent regressions and manual penetration testing remains essential for logic-level flaws". P1 stated "we rely on edge-case analysis and scenario-based testing because linters can't catch logic flaws", suggesting that explicit modelling exposes paths that scanners do not find. Selenium [13] can also be utilized in this stage, specifically in functional testing, by simulating realistic user interactions to detect logic flaws in web applications.

7. Integration and deployment: When a vulnerability is detected, it is necessary to clarify in advance which teams will be alerted and how, and to develop the subsequent action plan. By using AI in corporations, security personnel can respond more quickly to threats after they have been identified. When a cyber-attack is detected, AI sends an alarm to the information security control system. This enhances overall efficiencies in labor utilisation [51]. Nevertheless, P5 noted that tool blind-spots meant that manual reviews were still necessary in some instances.

8. Continuous learning and improvement: Establishing a living, evolving, system is arguably the most critical aspect of this 8-stage process. AI systems must remain relevant by adapting to changes in business processes and emerging security threats. Hofesh [5] (paragraph 15) notes that "continuous testing and automation is ..... the only way to ensure maximum security in defending against BLVs, at least at a high level". In this context, P4 noted that "elevated privileges expire after 30 min; we adjust after each incident". Additionally, the model used must be kept up to date, and the way it works should be reviewed periodically. Interviewees confirmed that they practice incremental tuning, embedding active-learning retraining loops in their maintenance procedures.

The interview data provided first-hand practitioner perspectives that support the value of the BLV framework outlined above. Practitioners' reliance on UML flow diagrams underscores the relevance of Stage 1 (Identify business logic components), while their frustrations with shallow logging highlights the importance of the data-engineering focus in Stage 2. High-impact but low-visibility flaws—race conditions, silent API truncations and fraud heuristics—map directly onto the anomaly-detection capability envisaged in Stages 5 and 6. Policy expiry windows and post-incident script updates reported by participants provide real-world evidence for embedding continuous-learning mechanisms into the on-going maintenance phase of the framework (Stage 8). Table 4 enhances the eight-stage framework by clarifying the process, objective and expected consequences at each stage in the cycle.

Table 4. AI-supported BLV detection framework: process, objectives and consequences at each stage.

Stage	Process	Objective	Consequences (Key Outcome)
1. Identify business logic components	Map all business processes and their interactions. Analyze source code using data/control flow analysis and annotated programs. Utilize NLP, Graph Analysis, and LLMs for analysis.	To understand the complete structure and flow of operations to identify potential points of failure.	A clear and comprehensive model of the application's intended logic (e.g., in UML diagrams), which serves as the "ground truth" for detecting future deviations.
2. Data collection & pre-processing	Collect historical data including logs, user interactions, and transaction records. Use tools like Selenium to simulate user behavior and Daikon to collect execution traces and infer program invariants.	To gather comprehensive data from diverse sources and establish a behavioral baseline of normal program execution.	A rich, structured dataset and a "behavioral fingerprint" of the application, addressing issues like insufficient logging and providing the foundation for model training.
3. AI model selection & training	Train AI models (e.g., Neural Networks, SVMs) using the collected data and records of known vulnerabilities. Use LLMs for program invariant prediction.	To train models on diverse data to accurately distinguish between legitimate transactions and potential vulnerabilities manually by Supervised ML or automatically using AutoML.	A trained, context-aware AI model capable of identifying known vulnerability patterns, aiming to reduce the false positives that practitioners report with existing tools.
4. Feature selection	Extract relevant features from data that indicate logic flaws. Use domain knowledge to select features that highlight anomalies in mission-critical processes. Utilize techniques like Decision Trees and SHAP.	To identify the most critical data characteristics (e.g., abnormal login frequency) that signal a potential vulnerability.	A refined set of key risk factors that allows the AI model to focus on the most important signals, improving detection accuracy and efficiency.
5. Anomaly detection	Employ unsupervised learning methods (e.g., Autoencoders, Isolation Forest) to find deviations from normal patterns. Use violations of Daikon-inferred invariants to flag anomalies. AutoML for anomaly dedection.	To identify unexpected system behaviors and previously unknown vulnerabilities that deviate from established business logic.	The detection of novel and hidden logic flaws, such as credit-card fraud that may have previously gone unnoticed by other systems.

Table 4. *Cont.*

Stage	Process	Objective	Consequences (Key Outcome)
6. Validation & testing	Test the model on unseen data to evaluate real-world performance. Use layered testing, including unit tests, manual penetration testing, and functional testing with Selenium. Employ adversarial testing techniques (e.g., FGSM, PGD).	To ensure the model accurately identifies true vulnerabilities while minimizing false positives and unnecessary alerts.	A robust, validated model with higher accuracy and lower false alarm rates, increasing trust in the automated detection system.
7. Integration & deployment	Integrate the model into the security control system, sending automated alerts upon threat detection. Have a pre-defined action plan for which teams will be alerted and how they will respond.	To deploy the validated model into the live environment and automate the initial stages of incident response.	Faster threat response times and enhanced labor utilization, though manual oversight may still be needed for certain cases.
8. Continuous learning & improvement	Establish a system that adapts to changing business processes and new threats. Use Reinforcement Learning and Active Learning to learn from feedback. Periodically retrain and review the model, embedding learning loops into maintenance.	To create a living, evolving defense system that maintains its relevance and effectiveness over time.	A resilient and adaptive security framework that can defend against emerging BLVs, reflecting the practice of continuous improvement seen in industry.

4. Discussion

The analysis of the interview material and development of the operational framework highlight a number of issues worthy of further discussion. Firstly, the analysis indicates that, at present, BLV detection remains largely reactive and resource intensive. Manual human insight is crucial for identifying BLVs, but this manual-first approach creates substantial bottlenecks due to limited quality assurance resources and time constraints. This issue is especially pronounced in startup environments, where testing is often sacrificed under delivery pressure. As P5 noted, "we discover many BLVs in production because we can't cover every edge path", highlighting the reactive nature of current detection practices. P4 suggested that "a hybrid approach—tools like Amazon Q complementing manual analysis—is most effective". Additionally, the phenomenon of developer familiarity bias—the tendency of developers to overlook flaws in their own code—indicates a need for external review mechanisms, such as peer reviews or red-teaming (disaster recovery simulation) exercises. Several participants also pointed out the limitations of automated tools, maintaining that automated scanners lack business-context understanding. This underscores a significant technological gap in current application security tools.

Secondly, BLVs are contextual, subtle, and have high impact. The findings reveal that BLVs are not always caused by traditional technical bugs; instead, they often arise from misunderstood or misimplemented business rules. These issues tend to present themselves as subtle misconfigurations, timing problems, or gaps in logic that may appear correct under normal conditions but fail in specific circumstances. For example, interviewees recorded how a single permission misconfiguration resulted in unauthorized access to sensitive data, while timing-related logic errors caused pricing discrepancies that directly impacted revenue. Race conditions were found to create "phantom" funds, allowing users to spend balances that did not actually exist. Additionally, weaknesses in fraud detection logic enabled stolen credit cards to be used without triggering any alerts, and undocumented

API limits led to silent data truncation, resulting in inaccurate outcomes. These examples collectively demonstrate that BLVs can have a significant business impact, despite being largely undetectable by conventional vulnerability detection tools. This highlights their context-dependent nature and emphasizes the need for domain-specific understanding—a capability that current automated scanning technologies often lack.

Thirdly, effective mitigation requires a hybrid, multi-layered approach. Effective management of BLVs relies on a combination of human expertise, domain modeling, and selective automation. Interviewees articulated a layered defence model that blends formal testing, domain modelling and selective automation. A common pattern starts with unit tests for regressions, moves to client-oriented user acceptance testing (UAT), and culminates in manual penetration testing focused on logic abuse. Additionally, workflow modeling and scenario-based testing were emphasized as being of particular value for revealing hidden logic paths. By simulating realistic user interactions to identify logic flaws in web applications, Selenium [13] can be useful for functional testing. Tools like Amazon Q were mentioned as helpful in identifying anomalies that require human verification, suggesting a promising human-in-the-loop model for future BLV detection systems. In stage 2 (data collection and pre-processing), program invariants can be found using tools like Daikon [14,42] or using LLM techniques [15]. Importantly, companies that adopted least privilege access policies with time-bound permissions demonstrated proactive risk management, showing that operational guardrails can limit the impact of undetected BLVs. Although no single practice dominates, the emerging best practice is clear: combine multi-stage human testing and explicit business-workflow modelling with lightweight, context-aware automation, thus exploiting the strengths of each while recognising the current limits of machine-only solutions. Program invariants are also useful for anomaly detection in BLV.

Fourthly, the detection framework put forward here can be used within and alongside the software development life cycle (SDLC). Each stage of the framework can be aligned with standard SDLC phases—from requirements gathering to software maintenance—and incorporated into each phase as appropriate. The integration of domain-specific modeling, layered testing strategies, and hybrid automation guided by human expertise enables early identification and mitigation of BLVs throughout the SDLC. Further, the interview data provided insights into how two modern software development approaches, low-code platforms and DevSecOps pipelines, affect the risks and opportunities for managing BLVs. Low-code platforms emerged in the interviews as a double-edged sword for BLV management as noted by Lethbridge [52]. Participants praised the speed at which visual, drag-and-drop workflows can be presented to clients and refined during UAT. Participant P6, for example, noted that their in-house low-code tool "allows us to move from needs analysis to a working prototype in just days, giving us adequate time for UAT before the release". However, this rapid development can conceal underlying complexities. Developers often introduce "nice-to-have" features on the fly, leading to undocumented paths that may compromise the integrity of business rules. P1, whose startup business integrates custom code into FlutterFlow (an AI app builder), admitted that after a few sprints, "the system becomes harder to control and debug". This concern was echoed by P5, who observed that low-code components generated by LLMs sometimes expose API keys or embed insecure defaults. Overall, these insights suggest that while low-code can shorten feedback cycles—an advantage for BLV discovery—it also necessitates the explicit export of visual flows into machine-readable graphs. This practice would enable logic-aware testing and anomaly detection, allowing for the identification of issues that might not be obvious to the naked eye.

The discussion around DevSecOps paints a complementary picture: while automation is widespread, its effectiveness often stops at syntax checks. Participants using continuous

integration-continuous delivery (CI/CD) pipelines reported having robust static scans and unit-test gates—as observed by Kushwaha et al. [53]—but acknowledged that these tools are generally insufficient for detecting BLVs. For example, even large Software-as-a-Service (SaaS) teams, utilizing hybrid services like Amazon Q, admitted that their focus tends to be more on infrastructure vulnerabilities rather than logic flaws. Despite this limitation, the DevSecOps culture that includes layered environments (e.g., Development, Test, Pre-Production, Production) remains valuable. As one participant pointed out, "different QA eyes at each stage increase the likelihood of detecting BLVs". The message is clear: while DevSecOps already provides the framework for automation, it lacks awareness of business context. By incorporating logic-specific test cases and policy checks as essential build gates—using workflow artifacts generated from low-code tools—organizations can effectively combine rapid delivery with reliable detection of BLVs.

Automated machine-learning (AutoML) frameworks, such as the open-source, distributed, in-memory H2O platform, simplify the processes of data preparation, al-gorithm selection, and hyperparameter tuning. By utilizing an AutoML framework in Stage 3 for model selection and training, practitioners can efficiently train and rapidly deploy high-performing predictive models with minimal manual effort. This approach not only reduces the occurrence of false positives but also eliminates the need for manual feature selection [46,47].

Figure 4 visually contrasts this advanced AutoML approach with the traditional machine learning workflow. In the traditional approach, depicted in the top loop, a human practitioner must manually perform each laborious step: data pre-processing, feature extraction and selection, model selection and tuning, and ensemble building. In contrast, the AutoML approach, shown in the bottom loop, automates these same complex and iterative steps, replacing the manual effort with a machine-driven process. The primary advantage illustrated is the significant reduction in time and specialised expertise required, allowing the practitioner to move from data collection to analyzing results more efficiently. This automation frees up human experts to focus on interpreting the final predictions and delivering business value, rather than on the time-consuming mechanics of model development.

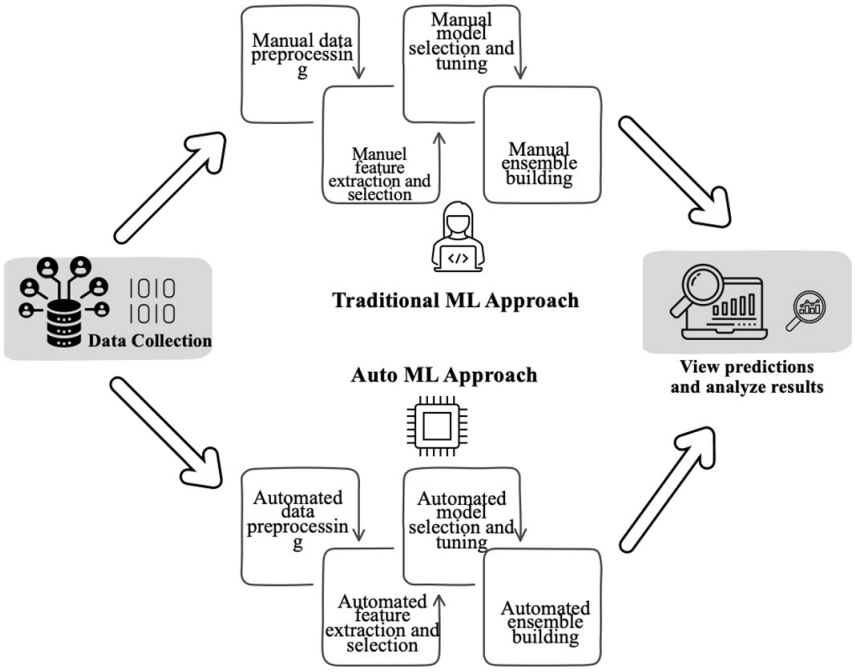

Figure 4. Comparison of the Traditional ML and AutoML approaches. Based on: [46].

Lastly, Mengi et al. [54] (p. 1) state that "Machine learning (ML) has become an important part of many aspects of our daily life. High-performance machine learning applications, on the other hand, necessitate the use of highly qualified data scientists and domain specialists. Automated machine learning (AutoML) aims to reduce the need for data scientists by allowing domain experts to automatically construct machine learning applications without extensive statistical and machine learning knowledge". This suggests that this concept is likely to increase in significance in the near future.

5. Conclusions

This study explores the topic of BLVs and sets out how AI may be deployed to enhance the detection and prevention of BLVs within wider organisational cybersecurity operations. From the existing literature, key AI-driven methodologies were identified, including machine learning-based anomaly detection, automated risk assessment, and proactive cybersecurity response mechanisms. In addition, the findings from the interviews provide a connection between practitioner experience and actionable strategy. They engender a comprehensive understanding of BLVs as complex, high-risk challenges that require a multi-dimensional response. Interview evidence allowed for the development of a more robust, AI-supported framework that can systematically detect and mitigate business logic flaws.

While this study provides an operational framework for AI-driven BLV detection, several limitations should be acknowledged. Although the development of the framework was supported by practitioner input, it has not yet been empirically tested and evaluated in real-world environments, requiring follow-on case study experimental implementations. Additionally, while the study explores AI and GenAI applications in cybersecurity, it does not compare the effectiveness of specific AI models or evaluate their performance in real-time security operations. A more in-depth analysis of machine learning techniques, their detection accuracy, and computational efficiency would further strengthen the framework's applicability. Moreover, the study does not address the potential risks of AI-based security solutions, such as false positives, adversarial attacks, or ethical concerns related to AI-driven automation.

Despite these limitations, the authors believe the study establishes a solid foundation for future research by highlighting AI's transformative role in business logic security. Addressing these challenges will help refine AI-driven cybersecurity strategies, ensuring scalable, adaptable, and resilient security frameworks for modern enterprises. The findings clearly indicate that traditional security tools often fail to address BLVs, which arise from flaws in the logical flow of business processes rather than conventional security weaknesses. The proposed framework provides a structured approach to overcoming these challenges. It integrates AI techniques for real-time monitoring, automated threat identification, and continuous learning-based cybersecurity defences. Future research could include application of the framework in different industries and organisational settings to assess its practical effectiveness and adaptability. The integration of the framework with software development methodologies would also be a profitable line of future enquiry, as BLVs often originate in the software development process. Quantitative studies on the costs and benefits of BLV detection would also prove of value and support the business case for required investment in BLV detection frameworks such as that put forward here.

The potential of AI and GenAI constitutes a transformational shift in cybersecurity, enabling businesses to move from reactive to proactive security strategies. Organisations can benefit strategically from investment in AI-driven cybersecurity architectures, incident response automation, and AI-enhanced security testing to safeguard business processes against evolving cyber threats. By fostering collaboration between cybersecurity professionals, AI researchers, and policymakers, more resilient and adaptive cybersecurity frameworks can be built that align with the requirements of modern business in the digital era.

Author Contributions: Conceptualisation, B.M.; methodology, B.M. and M.W.; validation, B.M. and M.W.; formal analysis, A.T. and Y.K.; investigation, B.M., A.T. and Y.K.; data curation, B.M., A.T., Y.K. and M.W.; writing—original draft preparation, B.M., A.T., Y.K. and M.W.; writing—review and editing, B.M. and M.W.; visualisation, B.M., A.T. and Y.K.; supervision, B.M.; project administration, B.M. and M.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study. The confidentiality and protection of the respondents' personal information are strictly guaranteed.

Data Availability Statement: The interview data referenced in this article is stored within a university environment. Enquiries can be made via the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Systematic Literature Review Detail

Table A1. Search strings used and sources located.

RQ	Search Terms	Count
	IEEE	
RQ1	("All Metadata":"Business Logic Vulnerabilities")	1
RQ1	("All Metadata":"Logic Attacks")	10
RQ1	("All Metadata":Business Logic Vulnerabilities) AND ("All Metadata":AI)	0
RQ1	("All Metadata":Logic Attacks) AND ("All Metadata":AI)	188
RQ1	("All Metadata":AI) AND ("All Metadata":business process) AND ("All Metadata":cyber security)	115
RQ1	("All Metadata":Artificial intelligence) AND ("All Metadata":"process security")	39
RQ1	("All Metadata":Artificial intelligence) AND ("All Metadata":"business security")	14
RQ1	("All Metadata":Artificial intelligence) AND ("All Metadata":business process) AND ("All Metadata":compromise)	1
RQ2	("All Metadata":"Artificial intelligence") AND ("All Metadata":vulnerability scanner)	20
RQ2	("All Metadata":"Artificial intelligence") AND ("All Metadata":Framework) AND ("All Metadata":Business Security) AND ("All Metadata":Vulnerability Detection)	16
RQ2	("All Metadata":"Artificial intelligence") AND ("All Metadata":threat detection) AND ("All Metadata":Business Process)	89
	Scopus	
RQ1	ALL ("Business Logic Vulnerabilities")	20
RQ1	ALL ("Logic Attacks")	117
RQ1	ALL ("Business Logic Vulnerabilities") AND ALL ("AI")	0
RQ1	ALL ("Logic Attacks") AND ALL ("AI")	5
RQ1	ALL ("AI") AND ALL ("Business Process") AND ALL ("Cyber Security") AND ALL ("Corporate")	25
RQ1	ALL (Artificial Intelligence) AND ("Business Process Security") AND ("Business Vulnerability")	9
RQ1	ALL (Artificial Intelligence) AND ("Business Logic Vulnerability")	6
RQ1	ALL (Artificial intelligence) AND ALL ("business process") AND ALL ("Generative AI") AND ("Cybersecurity")	3
RQ2	ALL ("Artificial Intelligence") AND ("Vulnerability Scanner") AND ("Cybersecurity") AND ("Detection")	64
RQ2	ALL ("Artificial Intelligence") AND ("Framework") AND ("Business Security") AND ("Vulnerability")	23
RQ2	ALL ("Artificial Intelligence") AND ("Threat Detection") AND ("Business Process")	128

Table A1. *Cont.*

RQ	Search Terms	Count
	ScienceDirect	
RQ1	"Business Logic Vulnerabilities"	6
RQ1	"Logic Attacks"	68
RQ1	"Business Logic Vulnerabilities" AND "AI"	1
RQ1	"Logic Attacks" AND "AI"	14
RQ1	"Artificial Intelligence" AND "Business Process" AND "Cyber Security" AND "Corporate Security"	6
RQ1	"Artificial Intelligence" AND "Business Vulnerability"	10
RQ1	"Artificial Intelligence" AND "Business Logic Vulnerability"	1
RQ1	Artificial intelligence AND "business process" AND "Generative AI" AND "Cybersecurity"	27
RQ2	"Artificial Intelligence" AND "Vulnerability Scanner" AND "Cybersecurity"	42
RQ2	"Artificial Intelligence" AND "Threat Detection" AND "Business Process"	55
RQ2	Artificial Intelligence AND "Framework" AND "Business Security" AND "Vulnerability"	29

Table A2. Business Logic Vulnerability related definitions from the Systematic Literature Review.

Li et al. [33]	"The business logic in the software design phase reflects the interaction between the objects. Such interactions may be exploited by an attacker, which can be used to break software system for illegitimate interests."
Stergiopoulos et al. [8]	"Application Business Logic Vulnerability" (BLV) is the flaw present in the faulty implementation of business logic rules within the application code."
Stergiopoulos et al. [8]	"Business logic vulnerabilities are an important class of defects that are the result of faulty application logic. Business logic refers to requirements implemented in algorithms that reflect the intended functionality of an application."
Pellegrino and Balzarotti [26]	"Logic vulnerabilities still lack a formal definition, but, in general, they are often the consequence of an insufficient validation of the business process of a web application."
Deepa & Thilagam [2]	"Business Logic Vulnerabilities (BLVs) are weaknesses that commonly allow attackers to manipulate the business logic of an application. They are easily exploitable, and the attacks exploiting BLVs are legitimate application transactions used to carry out an undesirable operation that is not part of normal business practice."
Ghorbanzadeh and Shahriari [11]	"Logic vulnerabilities are due to defects in the application logic implementation such that the application logic is not the logic that was expected. Indeed, such vulnerabilities pattern depends on the design and business logic of the application. There are no specific and common patterns for application logic vulnerabilities in commercial applications."

Table A2. *Cont.*

Kim et al. [28]	"Business logic vulnerabilities occur when the application logic is exposed to the client-side, allowing attackers to tamper with the business flow and perform unintended operations."
Zeller et al. [7]	"An important class of security problems are vulnerabilities in business rules. (. . .) Such attacks are called logical attacks and pose a distinct challenge to securing software applications. In logical attacks, weaknesses in the business rules are identified and exploited with the intent of disrupting services offered to legitimate users."
OWASP [6]	"Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses."
PortSwigger [27]	"Business logic vulnerabilities are flaws in the design and implementation of an application that allow an attacker to elicit unintended behavior. This potentially enables attackers to manipulate legitimate functionality to achieve a malicious goal. These flaws are generally the result of failing to anticipate unusual application states that may occur and, consequently, failing to handle them safely."

References

1. SiteLock. Statistics & Insights into Today's Most Challenging Cybersecurity Threats. 2021. Available online: https://www.sitelock.com/resources/security-report/ (accessed on 23 April 2025).
2. Deepa, G.; Thilagam, P.S. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Inf. Softw. Technol.* **2016**, *74*, 160–180. [CrossRef]
3. Affia, A. A Black-Box Methodology for Attacking Business Logic Vulnerabilities in Web Applications. Research Paper. Faculty of Information Technology, Tallinn University of Technology. 2017. Available online: https://www.researchgate.net/publication/336367792_A_Black-box_Methodology_for_Attacking_Business_Logic_Vulnerabilities_in_Web_Applications (accessed on 23 April 2025).
4. HackerOne. The HackerOne Top 10 Vulnerability Types. 2023. Available online: https://www.hackerone.com/lp/top-ten-vulnerabilities (accessed on 5 May 2025).
5. Hofesh, B. Business Logic Vulnerabilities: Busting the Automation Myth. 2025. Available online: https://brightsec.com/blog/business-logic-vulnerabilities-busting-the-automation-myth/ (accessed on 10 April 2025).
6. OWASP Foundation. Introduction to Business Logic. Available online: https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/10-Business_Logic_Testing/00-Introduction_to_Business_Logic (accessed on 10 May 2025).
7. Zeller, S.; Khakpour, N.; Weyns, D.; Deogun, D. Self-protection against business logic vulnerabilities. In Proceedings of the 2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Seoul, Republic of Korea, 25–26 May 2020; pp. 174–180.
8. Stergiopoulos, G.; Tsoumas, B.; Gritsalis, D. On business logic vulnerabilities hunting: The APP_LogGIC framework. In *Network and System Security: Proceedings of the 7th International Conference, NSS 2013, Madrid, Spain, 3–4 June 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 236–249.
9. Ramezan, C.A. Examining the cyber skills gap: An analysis of cybersecurity positions by sub-field. *J. Inf. Syst. Educ.* **2023**, *34*, 94–105.

10. Sheikh, A. Introduction to Ethical Hacking. In *Certified Ethical Hacker (CEH) Preparation Guide: Lesson-Based Review of Ethical Hacking and Penetration Testing*; Apress: Berkeley, CA, USA, 2021; pp. 1–9.
11. Ghorbansadeh, M.; Shahriari, H.R. Detecting application logic vulnerabilities via finding incompatibility between application design and implementation. *IET Softw.* **2020**, *14*, 377–388. [CrossRef]
12. OWASP. Web Security Testing Guide. 2021. Available online: https://owasp.org/www-project-web-security-testing-guide/ (accessed on 21 February 2025).
13. Selenium. Selenium WebDriver. 2025. Available online: https://www.selenium.dev/documentation/webdriver/ (accessed on 30 May 2025).
14. Felmetsger, V.; Cavedon, L.; Kruegel, C.; Vigna, G. Toward automated detection of logic vulnerabilities in web applications. In Proceedings of the 19th USENIX Security Symposium (USENIX Security 10), Washington, DC, USA, 11–13 August 2010.
15. Pei, K.; Bieber, D.; Shi, K.; Sutton, C.; Yin, P. Can large language models reason about program invariants? In Proceedings of the 40th International Conference on Machine Learning, Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 27496–27520. Available online: https://proceedings.mlr.press/v202/pei23a.html (accessed on 15 May 2025).
16. Maslak, O.I.; Maslak, M.V.; Grishko, N.Y.; Hlazunova, O.O.; Pererva, P.G.; Yakovenko, Y.Y. Artificial intelligence as a key driver of business operations transformation in the conditions of the digital economy. In Proceedings of the 2021 IEEE International Conference on Modern Electrical and Energy Systems (MEES), Kremenchuk, Ukraine, 21–24 September 2021; pp. 1–5. [CrossRef]
17. Rawindaran, N.; Jayal, A.; Prakash, E. Machine Learning Cybersecurity Adoption in Small and Medium Enterprises in Developed Countries. *Computers* **2021**, *10*, 150. [CrossRef]
18. Zhao, Z.C.; Li, D.X.; Dai, W.S. Machine-learning-enabled intelligence computing for crisis management in small and medium-sized enterprises (SMEs). *Technol. Forecast. Soc. Change* **2023**, *191*, 122492. [CrossRef]
19. Cubric, M.; Li, F. Bridging the 'Concept-Product' gap in new product development: Emerging insights from the application of artificial intelligence in FinTech SMEs. *Technovation* **2024**, *134*, 103017. [CrossRef]
20. Neupane, S.; Fernandez, I.A.; Mittal, S.; Rahimi, S. Impacts and Risk of Generative AI Technology on Cyber Defense. 2023. Available online: https://arxiv.org/pdf/2306.13033 (accessed on 14 February 2025).
21. Metin, B.; Özhan, F.G.; Wynn, M. Digitalisation and Cybersecurity: Towards an Operational Framework. *Electronics* **2024**, *13*, 4226. [CrossRef]
22. Kitchenham, B.; Charters, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*; Keele University: Keele, UK; University of Durham: Durham, UK, 2007; Volume 2.
23. Braun, V.; Clarke, V. Using thematic analysis in psychology. *Qual. Res. Psychol.* **2006**, *3*, 77–101. [CrossRef]
24. Guest, G.; Bunce, A.; Johnson, L. How Many Interviews Are Enough? An Experiment with Data Saturation and Variability. *Field Methods* **2006**, *18*, 59–82. [CrossRef]
25. Flick, U. *An Introduction to Qualitative Research*, 4th ed.; Sage Publications, Ltd.: London, UK, 2009.
26. Pellegrino, G.; Balsarotti, D. Toward Black-Box Detection of Logic Flaws in Web Applications. In Proceedings of the NDSS Symposium 2014, San Diego, CA, USA, 23–26 February 2014; Volume 14, pp. 23–26.
27. PortSwigger. Business Logic Vulnerabilities. Available online: https://portswigger.net/web-security/logic-flaws#what-are-business-logic-vulnerabilities (accessed on 7 April 2025).
28. Kim, I.L.; Zheng, Y.; Park, H.; Wang, W.; You, W.; Aafer, Y.; Shang, X. Finding client-side business flow tampering vulnerabilities. In Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), Seoul, Republic of Korea, 5–11 October 2020; pp. 222–233.
29. Alidoosti, M.; Nowroosi, A.; Nickabadi, A. Evaluating the web-application resiliency to business-layer DoS attacks. *ETRI J.* **2020**, *42*, 433–445. [CrossRef]
30. Alidoosti, M.; Nowroosi, A.; Nickabadi, A. BLProM: A Black-Box Approach for Detecting Business-Layer Processes in the Web Applications. *J. Comput. Secur.* **2019**, *6*, 65–80.
31. Alidoosti, M.; Nowroosi, A.; Nickabadi, A. Business-Layer Session Pussling Racer: Dynamic Security Testing Against Session Pussling Race Conditions in Business Layer. *ISC Int. J. Inf. Secur.* **2022**, *14*, 83–104.
32. Li, X.; Xue, Y. LogicScope: Automatic discovery of logic vulnerabilities within web applications. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, Hangzhou, China, 8–10 May 2013; pp. 481–486.
33. Li, X.; Meng, G.; Feng, S.; Li, X.; Pan, D. A framework based security-knowledge database for vulnerabilities detection of business logic. In Proceedings of the 2010 International Conference on Optics, Photonics and Energy Engineering (OPEE), Wuhan, China, 10–11 May 2010; Volume 1, pp. 292–297.
34. Deepa, G.; Thilagam, P.S.; Praseed, A.; Pais, A.R. DetLogic: A black-box approach for detecting logic vulnerabilities in web applications. *J. Netw. Comput. Appl.* **2018**, *109*, 89–109. [CrossRef]
35. Taubenberger, S.; Jürjens, J.; Yu, Y.; Nuseibeh, B. Resolving vulnerability identification errors using security requirements on business process models. *Inf. Manag. Comput. Secur.* **2023**, *21*, 202–223. [CrossRef]

36. Merrell, S.A.; Stevens, J.F. Improving the vulnerability management process. *EDPAC: EDP Audit Control Secur. Newsl.* **2008**, *38*, 13–22. [CrossRef]
37. Al-Turkistani, H.F.; Aldobaian, S.; Latif, R. Enterprise architecture frameworks assessment: Capabilities, cyber security and resiliency review. In Proceedings of the 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA), Riyadh, Saudi Arabia, 6–7 April 2021; pp. 79–84.
38. Chalvatsis, I.; Karras, D.; Papademetriou, R. Evaluation of Security Vulnerability Scanners for Small and Medium Enterprises Business Networks Resilience towards Risk Assessment. In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 29–31 March 2019; pp. 52–58.
39. McMahon, E.; Patton, M.; Samtani, S.; Chen, H. Benchmarking Vulnerability Assessment Tools for Enhanced Cyber-Physical System (CPS) Resiliency. In Proceedings of the 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), Miami, FL, USA, 9–11 November 2018; pp. 100–105.
40. Altaha, S.; Rahmann, M.M. A Mini Literature Review on Integrating Cybersecurity for Business Continuity. In Proceedings of the 2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Bali, Indonesia, 20–23 February 2023; pp. 353–359. [CrossRef]
41. Lópes, M.T.G.; Gasca, R.M.; Péres-Álvares, J.M. Compliance validation and diagnosis of business data constraints in business processes at runtime. *Inf. Syst.* **2015**, *48*, 26–43.
42. Ernst, M.D.; Perkins, J.H.; Guo, P.J.; McCamant, S.; Pacheco, C.; Tschantz, M.S.; Xiao, C. The Daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.* **2007**, *69*, 35–45. [CrossRef]
43. Marin-Castro, H.; Tello-Leal, E. Event Log Preprocessing for Process Mining: A Review. *Appl. Sci.* **2021**, *11*, 10556. [CrossRef]
44. Fiore, U.; Santis, A.D.; Perla, F.; Sanetti, P.; Palmieri, F. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Inf. Sci.* **2017**, *479*, 448–455. [CrossRef]
45. Raychaudhuri, K.; Kumar, M.; Bhanu, S. A Comparative Study and Performance Analysis of Classification Techniques: Support Vector Machine, Neural Networks and Decision Trees. In Proceedings of the International Conference on Intelligent Computing, Communication and Convergence, Ghaziabad, India, 11–12 November 2016; pp. 13–21.
46. Gyimah, N.K.; Akinie, R.; Mwakalonge, J.; Izison, B.; Mukwaya, A.; Ruganuza, D.; Sulle, M. An AutoML-based approach for network intrusion detection. In Proceedings of the 2025 IEEE SoutheastCon, Concord, NC, USA, 22–30 March 2025; pp. 1177–1183. [CrossRef]
47. LeDell, E.; Poirier, S. H_2O AutoML: Scalable automatic machine learning. In Proceedings of the 7th ICML Workshop on Automated Machine Learning, Online, 17–18 July 2020; Available online: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf (accessed on 12 May 2025).
48. Sabeel, U.; Heydari, S.S.; Elgassar, K.; El-Khatib, K. Building an intrusion detection system to detect atypical cyberattack flows. *IEEE Access* **2021**, *9*, 94352–94370. [CrossRef]
49. Dasgupta, S.; Yelikar, B.V.; Naredla, S.; Ibrahim, R.K.; Alassam, M.B. AI-powered cybersecurity: Identifying threats in digital banking. In Proceedings of the 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 12–13 May 2023; pp. 2614–2619.
50. Tao, C.; Gao, J.; Wang, T. Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices. *IEEE Access* **2019**, *7*, 120164–120175. [CrossRef]
51. Raad, R.; Searah, S.A.; Saleem, M.; Al-Tahee, A.M.; Abbas, S.Q.; Kadhim, M. Research on Corporate Protection Systems using Advanced Protection Techniques and Information Security. In Proceedings of the 2023 International Conference on Emerging Research in Computational Science (ICERCS), Coimbatore, India, 7–9 December 2023; pp. 1–6.
52. Lethbridge, T.C. Low-code is often high-code, so we must design low-code platforms to enable proper software engineering. In *Leveraging Applications of Formal Methods, Verification and Validation: Proceedings of the 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, 17–29 October 2021*; Springer International Publishing: Cham, Switzerland, 2021; pp. 202–212.
53. Kushwaha, M.K.; David, P.; Suseela, G. Automation and DevSecOps: Streamlining Security Measures in Financial System. In Proceedings of the 2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 12–14 July 2024; pp. 1–6.
54. Mengi, G.; Singh, S.K.; Kumar, S.; Mahto, D.; Sharma, A. Automated machine learning (AutoML): The future of computational intelligence. In *International Conference on Cyber Security, Privacy and Networking (ICSPN 2022)*; Springer International Publishing: Cham, Switzerland, 2021; pp. 309–317.