Received: 15 July 2024 | Revised: 9 April 2025 | Accepted: 29 April 2025 | Published online: 23 May 2025

RESEARCH ARTICLE

Artificial Intelligence and Applications
2025, Vol. 3(4) 392–407
DOI: 10.47852/bonviewAIA52023841

SNERC: Enhancing Knowledge Management with Named Entity Recognition and Document Classification for Apply Gaming

Philippe Tamla[1,*], **Florian Freund**[1] **and Matthias Hemmje**[1]

[1]*Faculty of Multimedia and Computer Science, Hagen University, Germany*

Abstract: In this article, we present Stanford Named Entity Recognition and Classification (SNERC), an intelligent system designed to enhance knowledge management through named entity recognition (NER) and document classification (DC) in the field of Applied Gaming. In this domain, the effective application of NER and DC is essential for addressing information overload (IO), enabling software developers to efficiently search, filter, and retrieve large volumes of textual data from web sources. SNERC streamlines the management and deployment of machine learning (ML)-based NER models, supporting the accurate extraction of named entities (NEs) and the classification of heterogeneous textual documents. The system tackles key challenges in NER, such as the impact of language and domain specificity on model performance, domain adaptation, and the complexity of handling diverse NE types. We demonstrate SNERC's capabilities through real-world use cases, highlighting improvements in DC and information retrieval (IR) within applied gaming scenarios. The system provides core functionalities for training, evaluating, and managing NER models using the Stanford CoreNLP framework. Additionally, SNERC integrates with a rule-based expert system (RBES) to enable the automatic categorization of documents into predefined taxonomies within a knowledge management system. We present results from comprehensive qualitative and quantitative evaluations—measured through precision, recall, and F-score—to assess the system's effectiveness and identify areas for further optimization, supporting seamless integration into real-world operational environments.

Keywords: named entity recognition, document classification, rule-based expert system, social network, knowledge management system

1. Introduction

This paper addresses the persistent challenge of information overload (IO) encountered by software developers when accessing vast and unstructured textual resources on the Web. The goal is to facilitate structured and meaningful access to these documents using named entity recognition (NER) [1] and document classification (DC) [2], thereby improving the efficiency and effectiveness of information retrieval (IR). IO in the semantic web context impedes comprehension and decision-making due to the excessive availability of information. IR techniques such as searching and browsing help manage this overload, but their effectiveness can be significantly enhanced by incorporating NER and DC. NER, a key subfield of natural language processing (NLP), identifies named entities (NEs) in textual data, supporting semantic analysis, IR, and information discovery (ID). With advances in machine learning (ML), NER techniques have become more robust and are now widely used across domains such as software engineering, social media, and medical research. DC, another fundamental IR approach, organizes text documents into content-based categories. Automatic DC—using ML algorithms or rule-based expert systems (RBESs)—has become essential for managing the growing volume of digital documents. This research is motivated by two Research and Development (R&D) projects: "Realising an Applied Gaming Ecosystem" (RAGE) and the DFG-funded "Recommendation Rationalisation" (RecomRatio) project [3]. RAGE supports Serious Games Development (SGD) by providing reusable software components via the portal gamecomponent.eu, which remains in active use. These components support analytics, avatar systems, emotion detection, gamification, NLP, and personalization. Serious Games (SGs), also referred to as Applied Games [4], are developed not only for entertainment but also for educational, training, and motivational purposes [5]. Their usage has expanded in education, industry, and research over the last decade [6, 7]. Corporations increasingly invest in SGs to enhance business functions [8], as evidenced by IBM's endorsement of their potential for collaborative problem-solving, real-time modeling, and process improvement [9]. However, SGD remains complex, involving game design, software architecture, content creation, and testing [3]. Its success depends on high-quality gamification platforms and reusable software assets [10]. To support these developments, the RAGE portal integrates the Knowledge Management-Ecosystem Portal (KM-EP), developed by the Chair of Multimedia and Internet Applications at the University of Hagen and FTK e.V. [11]. KM-EP supports knowledge management across domains [3] and has been employed in H2020 projects such as Metaplat, SenseCare, and RecomRatio. Through a taxonomy management system (TMS),

*Corresponding author: Philippe Tamla, Faculty of Multimedia and Computer Science, Hagen University, Germany. Email: philippe.tamla@fernuni-hagen.de

KM-EP enables hierarchical classification of textual content. Moreover, the portal integrates with social and collaborative platforms like Stack Overflow and GitHub, allowing developers to import and classify user-generated content via a REST API [3]. Faceted search capabilities assist in navigating large document repositories. However, manual classification remains inefficient, indicating a clear need for automated NER and DC. Similarly, RecomRatio targets evidence extraction from medical texts, emphasizing the handling of emerging named entities for medical decision support [3]. This paper presents the development of SNERC (Stanford Named Entity Recognition and Classification), a system that supports both novice and expert users in applying NER and DC to enhance IR within knowledge management systems (KMS) such as KM-EP.

The study is guided by three Research Questions (RQs). *RQ1 investigates how a NER and DC system can be developed to support novice developers in accessing textual resources*. This involves reviewing core IR concepts, analyzing existing NER/DC systems, and identifying optimal ML-based solutions that integrate NLP for effective classification and retrieval. *RQ2 focuses on how to design a scalable model for NE extraction and DC.* It entails studying unstructured web content accessed by users, training ML models for domain-specific text, and enabling seamless integration of extracted NEs into external systems such as the RAGE portal. *RQ3 explores how the system can support faceted search and browsing within a KMS.* This question addresses leveraging extracted NEs and taxonomy hierarchies to enable intelligent document navigation, requiring a review of semantic analysis, indexing, and content reasoning technologies.

The research is framed using the research methodology of Nunamaker et al. [12], which offers a structured approach encompassing observation, theory building, system development, and experimentation. This methodology ensures a rigorous evaluation of the proposed system in real-world scenarios. Accordingly, the research objectives include *reviewing* literature on IO, IR, NER, and DC; *analyzing* SGD-relevant content from social platforms; *designing and developing* a robust NER and DC model; investigating integration technologies within KM-EP; and *implementing and evaluating* the system's usability and feasibility. The structure of the paper is as follows: the introduction outlines the motivation, RQs, methodology, and objectives; the second chapter reviews the state of the art; the third chapter models our system; the fourth chapter details system implementation; the fifth presents evaluation results of SNERC; and the final chapter concludes with insights and future directions. This introduction sets the foundation for exploring the technical, methodological, and practical challenges of enhancing IR through NER and DC in complex, information-rich domains such as SGs and medical research.

2. State of the Art and Related Work

2.1. RAGE

As previously noted, the RAGE social platform supports the integration of diverse web-based textual resources, such as Stack Exchange discussions, into its knowledge management ecosystem. These documents typically include a title, description, and metadata —such as tags and keywords—allowing for structured organization. A key component of RAGE is its integrated TMS, which enables the classification of documents into hierarchical taxonomies within a **KMS**. KMSs are specialized systems for managing and leveraging organizational knowledge assets [13]. Among them, the KM-EP is a versatile and cross-domain solution built as a PHP-based web portal, providing researchers with a flexible framework to deploy their work without implementing foundational services like data persistence or user management [14]. *Taxonomy*, the science of classification [15], is leveraged in RAGE to support faceted browsing, enabling users to refine search results and improve IR efficiency. However, the manual classification of documents into multiple hierarchical taxonomies presents significant challenges, particularly due to the risk of confusion between taxonomies and the cognitive effort required to analyze and classify each document [14]. This process becomes increasingly time-consuming when scaling to large datasets. To address these limitations, the integration of a NER system into the RAGE platform is proposed. Such a system would automate the extraction of NEs from imported content and, when coupled with an RBES, enable intelligent reasoning over-extracted entities and their alignment with existing taxonomies, streamlining the classification process.

2.2. NER

Early research categorizes NER techniques broadly into rule-based and ML-based approaches [1]. Among the ML-based techniques, **Conditional Random Fields (CRFs)** are prominent for their effectiveness in sequence labeling tasks such as **part-of-speech (POS)** tagging, sentence segmentation, and NER [16–18]. NER system performance is commonly assessed using *precision*, *recall*, and *F1-score* [19]. Precision represents the ratio of correctly identified NEs to all identified NEs, while recall reflects the ratio of correctly identified NEs to all relevant NEs in the dataset. The $F1$-score balances both metrics and is widely used to evaluate NER systems. In our previous review [14], we identified six pertinent challenges—referred to as Remaining Challenges (RCs)—in the development of effective NER systems. RC1 relates to the high sensitivity of NER performance to the language and domain in which the system is applied. This is due to early systems being rule-based and language-specific, limiting their portability and generalizability [1]. RC2 concerns domain adaptation in ML-based approaches. Applying a model trained in one domain (e.g., clinical) to another (e.g., bioinformatics) often results in performance degradation [1]. To address this, researchers have focused on designing domain- and language-independent features to reduce training effort and improve transferability [1, 20]. RC3 highlights the challenge of recognizing complex NE types. While simple patterns may capture numeric dates, more nuanced entities—such as relative dates or domain-specific expressions—require sophisticated extraction strategies and external resources like gazetteers [1, 21]. RC4 addresses the complexity of feature selection in ML-based systems. As noted by Benajiba et al. [20], choosing an optimal set of features remains one of the most challenging aspects of NLP. Effective NER often requires combining local features (e.g., orthographic cues, lemmas) with broader contextual features (e.g., POS tags, n-grams, external knowledge bases) [22]. RC5 refers to the scarcity of domain-specific, gold-standard annotated datasets. Creating such resources is labor-intensive and demands both linguistic and subject-matter expertise—particularly in interdisciplinary areas like medicine [3]. RC6 focuses on preprocessing stages like tokenization, lemmatization, stemming, and annotation. If poorly executed, these steps can compromise model performance. Visualization tools are recommended to ensure quality control throughout the pipeline [3, 22]. In the context of SGD, implementing NER requires processing documents from Question-Answering Systems and Collaborative Development Environments, which often contain domain-specific NEs. Our formative study on NER in SGs [14] showed that systems in this space must handle naming variations, synonyms, and frequent misspellings. When evaluating available NER toolkits [14], we prioritized criteria such as domain independence, advanced NLP features (e.g., POS tagging, sentiment analysis, dependency parsing), open-source availability, and active community support. Based on this evaluation, Stanford CoreNLP was selected as the most suitable platform for implementing our prototype.

2.3. DC

DC refers to the task of assigning one or more categories to a document based on its semantic content [23]. Alongside NER, DC plays a pivotal role in enhancing IR by supporting both search and browsing functionalities [24]. In platforms such as KM-EP, DC facilitates the automatic organization of documents into topical categories, improving accessibility and knowledge management [3]. A major challenge in DC is identifying a document's subject matter—a task central to automatic content analysis [25]. While human understanding relies on contextual reasoning, computational systems must derive meaning through semantic and statistical techniques. Luhn [26] proposed that the frequency and co-occurrence of terms (e.g., "science," "mathematics," "learning") can signal a document's thematic focus. Sebastiani [27] identifies two principal approaches to DC: ML-based and rule-based classification. ML-based methods are typically implemented through supervised learning, where labeled training data are used to develop a classifier that maps document features to categories [23, 28]. Unsupervised learning, in contrast, clusters documents into groups based on textual patterns without the need for labeled data [29]. Although widely adopted, ML-based DC continues to face challenges in real-world applications, particularly the lack of high-quality, domain-specific training data and the limited generalizability of classifiers across different domains and languages [30–32]. As Kim et al. [30] observed, "*classification by machine learning may not keep the same performance because the knowledge generated from the training set may not be appropriate for certain types of web information*." To address these limitations, rule-based and hybrid classification approaches are increasingly used [30, 33, 34]. Rule-based systems rely on manually crafted rules informed by syntactic and semantic cues. These rules are often developed using NLP techniques such as **Syntactic Pattern Matching (SPM)**, POS tagging, and dependency parsing [35]. For example, Biswas et al. [33] applied rule-based NLP and SPM techniques to classify large web-based corpora. POS tagging plays a fundamental role in linguistic analysis and is widely applied in various NLP applications, including syntactic and semantic parsing, as well as machine translation [16]. It assigns syntactic labels—such as nouns, verbs, and adjectives—to words, helping extract sentence components like subject, predicate, and object [36, 37]. This grammatical structure supports the formulation of semantic classification rules based on sentence mood, polarity, and composition. Linguistic and grammatical analysis complements NLP by encoding textual data into structured formats that capture keyword relationships and semantic roles [38]. For instance, in POS-tagged text, the presence of particular NEs like "Java" or "C#" can indicate that the document belongs to a "Programming Language" category. Based on these insights, we identify three critical RCs for implementing rule-based DC. RC7 pertains to identifying recurring linguistic patterns and terms, which is often achieved using **Regular Expressions (REs)**, a technique widely applied in modern IR systems. RC8 concerns the recognition of domain-specific NEs and their semantic relationships. These can be identified and disambiguated more effectively by integrating NER and POS tagging systems. RC9 addresses the complexity of extracting well-formed natural language sentences and understanding their grammatical structure, which can be supported by visualization tools that highlight syntactic and semantic components (e.g., subject–predicate–object relationships). Since NEs often encode domain-specific knowledge, analyzing their interrelationships with surrounding text provides strong cues for classification.

To build an effective rule-based DC system, it is necessary to integrate REs, NER, POS tagging, and visualization tools to facilitate semantic analysis. These technologies enable the formulation of classification rules that reflect the reasoning patterns of human experts. A common implementation approach is the use of an RBES, which provides a human-readable interface for defining logical rules capable of achieving expert-level classification accuracy [34, 39]. The following section discusses RBES technologies and their architectural components.

2.3.1. RBESs

RBESs are increasingly adopting Artificial Intelligence (AI) technologies that simulate expert-level reasoning by encoding domain-specific knowledge in the form of logical rules. They are widely used in domains such as healthcare, science, engineering, and business [40–42]. An RBES represents knowledge acquired from human experts in a structured format—typically as symbolic or numerical rules—within a computer program [43]. According to Talukdar et al. [39], rules offer an expressive, straightforward, and flexible mechanism to represent and apply such knowledge. Negnevitsky [44] emphasizes that experts possess deep, experience-based knowledge which, when formalized, becomes a powerful basis for computational reasoning. A typical rule in an RBES consists of two parts: the antecedent, also known as the condition or premise, and the consequent, which specifies the action or conclusion. This logical relationship is expressed in a WHEN . . . THEN format, as illustrated in Listing 1:

Listing 1
Example rule with WHEN . . . THEN statements

```
WHEN <antecedent>
THEN <consequent>
```

A rule "fires" when the conditions in the antecedent block are satisfied. Rules may contain multiple antecedents connected using logical operators such as AND or OR, enabling complex reasoning chains. This is demonstrated in Listing 2, where multiple conditions are combined:

Listing 2
Example rule with antecedents

```
WHEN
<antecedent 1> WHEN <antecedent 1>
AND
<antecedent 2> OR <antecedent 2>
AND
<antecedent n> OR <antecedent n>
THEN
<consequent> THEN <consequent>
```

Each antecedent typically includes a linguistic object and a value, connected by an operator. Operators may be logical (e.g., is, is not) or mathematical (e.g., <, >), depending on whether symbolic or numeric values are being assigned. As shown in

Listing 3, mathematical constraints can define object properties to determine rule outcomes:

Listing 3
Example rule with mathematical operators

```
WHEN
a.height > “5cm”
AND
a.height < “3m”
AND
a is not Animal
THEN
a = person
```

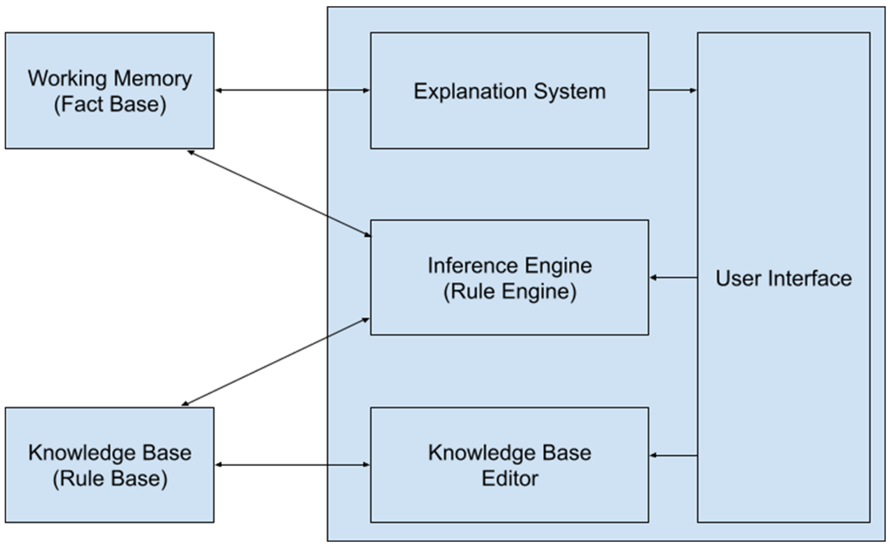
The core processing in an RBES is performed by the inference engine, which compares the rules stored in the knowledge base with the facts held in the working memory. When the antecedent of a rule matches the current facts, the rule fires, and its corresponding action is executed. This may result in new facts being added to the working memory, extending the reasoning process through what is known as an inference chain [39]. Two main inference strategies are used: forward chaining and backward chaining. Listing 3 illustrates an example of a forward-chaining rule, where conditions are applied to factual data (e.g., an object’s height and classification) to derive a new fact—in this case, identifying the object as a person if certain height constraints are met and it is not an animal. Forward chaining follows a data-driven approach, where reasoning begins with known facts and applies rules to derive new knowledge. For example, the system Dendral uses forward chaining to determine molecular structures of unknown soil samples [45]. In contrast, backward chaining is goal-driven. It begins with a hypothesis or desired conclusion and works backward to verify whether existing rules and facts support it. The expert system MYCIN, which diagnoses infectious blood diseases, is a classical implementation of this technique [46].

This reasoning process is supported by a modular architecture, as depicted in Figure 1: Architecture of an RBES [47]. The knowledge base stores domain knowledge in the form of rules (i.e., WHEN . . . THEN structures). The working memory (fact base) maintains the dynamic set of facts during runtime, which are compared against rule conditions. The inference engine evaluates the rules and determines whether they fire based on the facts available. The explanation system enhances transparency by justifying the system’s reasoning and outputs to the user. The user interface facilitates interaction between the system and the user, while the knowledge base editor allows for the creation, modification, and management of rules within the system. This separation between the knowledge base and inference engine is one of the major benefits of RBESs, as it allows updates to the rule logic without affecting the reasoning engine. RBESs are also capable of handling incomplete or uncertain knowledge by assigning confidence levels or certainty factors to rules [48, 49], further improving their adaptability and robustness. To select an appropriate RBES tool for DC, we apply the same evaluation criteria used in our earlier comparison of NER frameworks [14]. The system must be open source to support academic and collaborative development. It must support both forward and backward chaining and allow the knowledge base to remain independent from the reasoning engine, ideally supporting object-oriented programming environments such as Java. Additionally, the system should demonstrate maturity and robustness—evidenced by active development, strong documentation, and wide adoption in academic or industrial contexts. Finally, the system must support DC tasks with manageable complexity, allowing users to define classification rules through a low learning curve while still enabling expert-level reasoning accuracy. The following section presents a comparative analysis of RBES implementations based on these criteria.

2.3.2. Comparison of RBES

One of the earliest examples of an RBES is MYCIN, a backward-chaining prototype developed in the 1970s to assist in diagnosing and recommending treatment for serious infections [50]. Its primary aim was to make the reasoning process transparent during consultations by displaying the rules under consideration at each step. MYCIN demonstrated the potential of encoding intelligent behavior in expert systems and contributed significantly to early

Figure 1
Architecture of an RBES


Working Memory (Fact Base)
Explanation System
Inference Engine (Rule Engine)
User Interface
Knowledge Base (Rule Base)
Knowledge Base Editor

AI research. However, like many AI programs of that era, its development was gradual and sometimes stagnant, ultimately preventing it from evolving into a robust and widely adopted RBES [46]. In contrast, the C Language Integrated Production System (CLIPS), developed by NASA in 1985, quickly gained popularity due to its efficiency and portability [51]. CLIPS supports forward chaining and was initially implemented in C, with a syntax similar to Lisp. It later integrated COOL (CLIPS Object-Oriented Language), enabling developers to combine procedural, object-oriented, and logical programming styles. While CLIPS offers a clear separation between the knowledge base and the inference engine, its syntax is comparatively more complex and less intuitive than newer systems. Since 1996, CLIPS has been available as public domain software; however, community support is limited, and its documentation lacks comprehensive updates [51]. Ten years after CLIPS, the Java Expert System Shell (JESS) was introduced by Friedman-Hill [52] at Sandia National Laboratories. JESS is a Java-based reimplementation of CLIPS, designed for tighter integration with Java applications. It retains compatibility with CLIPS' rule language but also supports JessML, a declarative XML-based syntax for rule specification. Although JESS is freely available for educational and governmental purposes, it is not open-source, and its source code is not publicly accessible under any license. A more recent and widely adopted system is Drools, an open-source RBES initially developed by Bob McWhirter in 2001 and later maintained by the JBoss community [53]. Written in Java, Drools uses WHEN . . . THEN rule syntax that is significantly easier to understand and more readable than that of CLIPS or JESS. Rule definitions in Drools can also be expressed in native XML format, offering additional flexibility. The system's inference engine is based on an extended version of the Rete algorithm, optimized for object-oriented programming environments. Drools is distributed under the Apache open-source license and is platform-independent, making it suitable for integration with Java-based systems. Its simplicity, flexibility, and extensibility have contributed to its adoption across diverse domains. A comparison of these RBESs is summarized in Table 1. Among the evaluated systems, Drools emerges as the most robust, with strong community support, wide adoption, and compatibility with both backward and forward chaining. Its low learning curve, readable syntax, and seamless integration with Java make it particularly well-suited for our context. Since Java is already used in the KM-EP platform and was selected for the development of our NER subsystem, Drools provides the ideal environment for implementing our DC subsystem. Its adoption ensures efficient integration with KM-EP and full compatibility with the sub-modules developed throughout this research.

2.4. SGs-related taxonomies

Taxonomies in SGs span multiple dimensions, each reflecting different aspects of game development and analysis. For the purposes of this research, the most relevant taxonomies are those related to (1) *game genre*, (2) *programming languages*, (3) *video game tools*, (4) *machine learning algorithms*, and (5) *video game specification* and *implementation bugs*. Numerous researchers have proposed hierarchical taxonomies in the SG domain with the primary objective of identifying key characteristics of popular SGs and offering structured frameworks for future research on their development and impact [54]. The first taxonomy adopted in our classification work is based on game genre [GEN], one of the foundational classification schemes for SGs [55]. A SG may be categorized according to its target market [GEN/MAR]—such as education, healthcare, or military —its game type [GEN/TYPE], including board games, card games, simulations, role-playing games, and toys, or by its deployment platform [GEN/PLA], such as browser, mobile, console, or PC [55]. These categories are frequently used in discussions on platforms like Stack Overflow, where developers use tags such as "education," "simulation," or "console." We therefore focus on classifying SG-related discussions according to this game genre taxonomy. The second taxonomy addresses programming languages [LANG], which is highly relevant given that SG developers often discuss the implementation of features using specific programming or scripting languages. This dimension is informed by van Roy [56], who proposes classifying languages according to their programming paradigms, such as object-oriented, functional, or logic-based paradigms. This classification enables a systematic approach to analyzing SG-related technical discussions centered around language-specific development concerns. Third, the tools and environments used in SG development represent another key dimension. Modern SGs often rely on well-established game engines and integrated development environments (IDEs). Toftedahl and Engström [57] propose a lightweight taxonomy that categorizes common game development tools into subcategories such as development environments [TOOL/IDE] and game engines [TOOL/ENG]. This taxonomy provides a standardized way to classify Stack Overflow posts that mention specific tools like Unity, Unreal Engine, or Visual Studio. Fourth, we consider the integration of ML, a fast-growing area in SG development. ML plays an increasingly central role in intelligent game systems and adaptive learning components,

Table 1
Comparison of RBESs

	MYCIN	CLIPS	JESS	Drools
License Supporting Features	GPL	Public-Domain	Public-Domain	ASL
backward chaining	yes	no	no	yes
forward chaining	no	yes	yes	yes
Separates Knowledge Base from Inference Logic	yes	yes	yes	yes
Object-oriented Programming	no	yes	no	yes
Robustness and Maturity				
Community Support	very weak	active	active	very active
Documentation Quality	bad	good	good	good
Supported Programming Languages	LIPS	C	Java	Java
Community Size	very small	medium	medium	large
User-friendly Rule Formulation	No	medium	Medium	yes
Learning Curve	steep	low	low	very low

particularly within reusable software modules [3]. For example, pedagogical systems now use observational gameplay data to improve adaptivity rather than relying on predefined theoretical models [58]. This trend justifies the inclusion of a ML taxonomy dimension in our classification model, enabling us to capture ML-related development concerns and innovations in SGs. The fifth and final dimension focuses on video game bugs [BUG], a recurring challenge for SG developers. Like traditional software engineers, SG developers frequently engage in troubleshooting and debugging tasks. Truelove et al. [59] propose a taxonomy that distinguishes between specification bugs [BUG/SPEC] and implementation bugs [BUG/IMP]. Specification bugs typically originate from flawed or incomplete design documents, including missing requirements, conflicting statements, or misinterpretations of user needs. In contrast, implementation bugs arise from assets—such as source code, art, or level design—produced to execute the specification. These bugs result in discrepancies between the intended and actual game behavior [59, 60]. By adopting these five taxonomies—game genre, programming languages, development tools, ML, and bugs—we establish a multidimensional framework for classifying SG-related discussions and artifacts. This framework supports more granular indexing, retrieval, and analysis of SG knowledge across platforms like Stack Overflow and contributes to the ongoing formalization of SG development practices.

2.5. Evaluation methodology

Various methods exist to evaluate whether research objectives have been met. Automated and formal techniques, such as testing systems against formal specifications using computer programs, are well-established [61]. However, due to their complexity, they are not applied in this study. Previously, we employed an informal evaluation method—Cognitive Walkthrough—which leverages the evaluators' experience and domain knowledge to identify usability issues [14, 62]. In this work, we shift toward empirical evaluation approaches, aiming to involve a broader range of users. Empirical methods are particularly effective in evaluating user interfaces, often producing high-quality results [63]. Among these, Focus Group Interviews enable early-stage feedback by conducting task-specific group discussions during the design phase [64]. The Thinking Aloud Method allows users to verbalize their thoughts and emotions while interacting with the system, providing insight into usability issues and optimization opportunities [65]. For quantifying system effectiveness and efficiency, Controlled Experiments are valuable. These require clearly defined tasks and goals, and the outcomes are evaluated using standard metrics such as Precision, Recall, and *F*1-Score [66]. Precision measures the accuracy of retrieved information, Recall assesses completeness, and F1 balances the two. To gather user feedback, surveys and questionnaires are widely used and have been recently applied in evaluating web-based KMSs [67]. For usability evaluation, standardized tools like UMUX (Usability Metric for User Experience) assess ease of use on a 7-point scale [68], while the USE Questionnaire measures usability, usefulness, satisfaction, and ease of learning [69]. To assess documentation quality, we apply the "Münsteraner Fragebogen zur Evaluation—Zusatzmodul Basistexte" [70]. Following a review of state-of-the-art methods and tools for NER and DC, the next section presents the design of our proposed system, based on the identified RCs.

3. The Proposed Methodology

The preceding chapter introduced key concepts and technologies that underpin our system's architecture for NER and DC. These components are part of the observation phase in our research methodology and serve to identify the central RCs in our work. In this chapter, we adopt the User-Centered Design (UCD) approach for system development. The term UCD was originally introduced by Wallach and Scholz [71] and later formalized by Norman and Draper [72] at the University of California, San Diego. UCD has been widely recognized for improving the usability and utility of software systems [73]. The process involves four iterative phases: (1) defining the use context (target users and their goals); (2) identifying user and business requirements; (3) generating design solutions; and (4) evaluating designs through usability testing and user feedback. For system modeling, we employ the unified modeling language, a standardized framework for documenting, specifying, and constructing software systems [74].

3.1. Use context

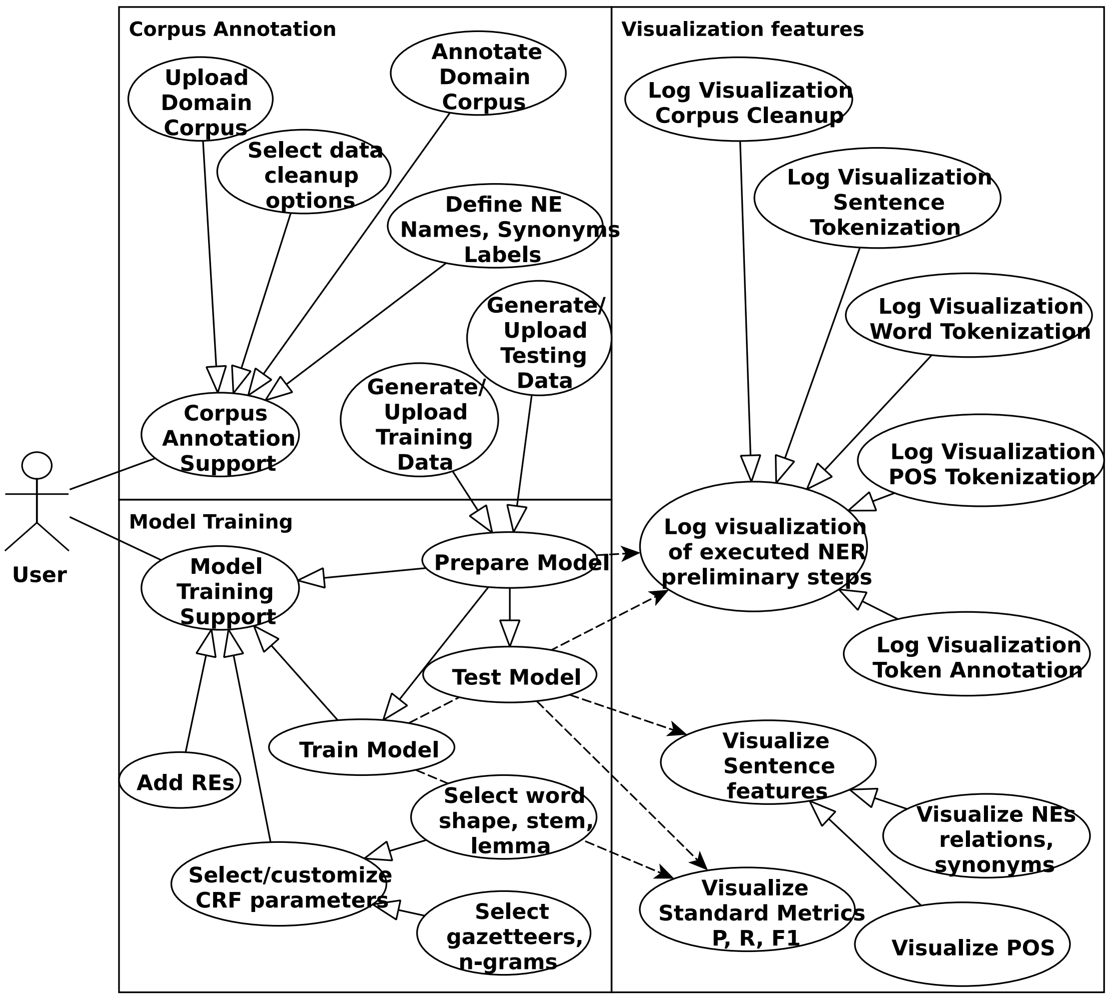
The user requirements for the NER component are grounded in the previously identified NER-related RCs (RC1–RC6) and our preparatory studies [14]. These highlight the need to develop functionalities enabling users—regardless of programming or ML expertise—to define and execute essential steps in a NER pipeline. These include selecting a domain-specific corpus, performing automatic annotation with custom NE labels, adjusting CRF parameters, and training/testing models. This addresses RC1, RC2, RC4, and RC5. The studies further emphasize that IR systems in dynamic environments like the Web must efficiently access SG-related resources—such as patterns, programs, or tools—through methods like NER and DC. To improve the gold standard for training data, features must also support defining synonyms and variations of a NE (RC3, RC5), as well as allow experts to revise automatically annotated data to mitigate overfitting during model training [75]. Since the quality of earlier NER steps significantly influences model performance (RC6), the system must include visualization features to display intermediate results and logs. These include metrics such as Precision, Recall, and F1, as well as NE labels, relationships between NEs, and their associated POS tags. For DC, the requirements stem from DC-related RCs (RC7–RC9). The system must support semantic rule-based classification via NER, POS tagging, and SPM. Users must be able to select document-level features—such as domain-specific NEs, POS tags, sentence structure, sentiment, and grammatical components—for rule formulation. The integration of a RBES allows for intuitive rule creation using WHEN . . . THEN constructs. Furthermore, to support context-aware filtering in IR systems [76], the DC component must enable rule-based classification using existing NER models and taxonomies within KM-EP. Visualization of semantic structures—including NEs, POSs, sentence forms, and document sentiment—is essential for identifying and refining rules and ensuring classification accuracy.

3.2. Use cases

Our use cases are designed in accordance with our research motivation (Section 1), the identified RCs (RC1–RC9), and the preparatory studies. The system must empower both novice and expert users to train new NER models and use them for extracting NEs from documents. These extracted entities serve as the basis for generating semantic rules, which are then used to classify documents into taxonomy categories in KM-EP.

The first use case, shown in Figure 2, supports NER within the KM-EP environment. Users can define parameters and configurations—collectively referred to as a NER model

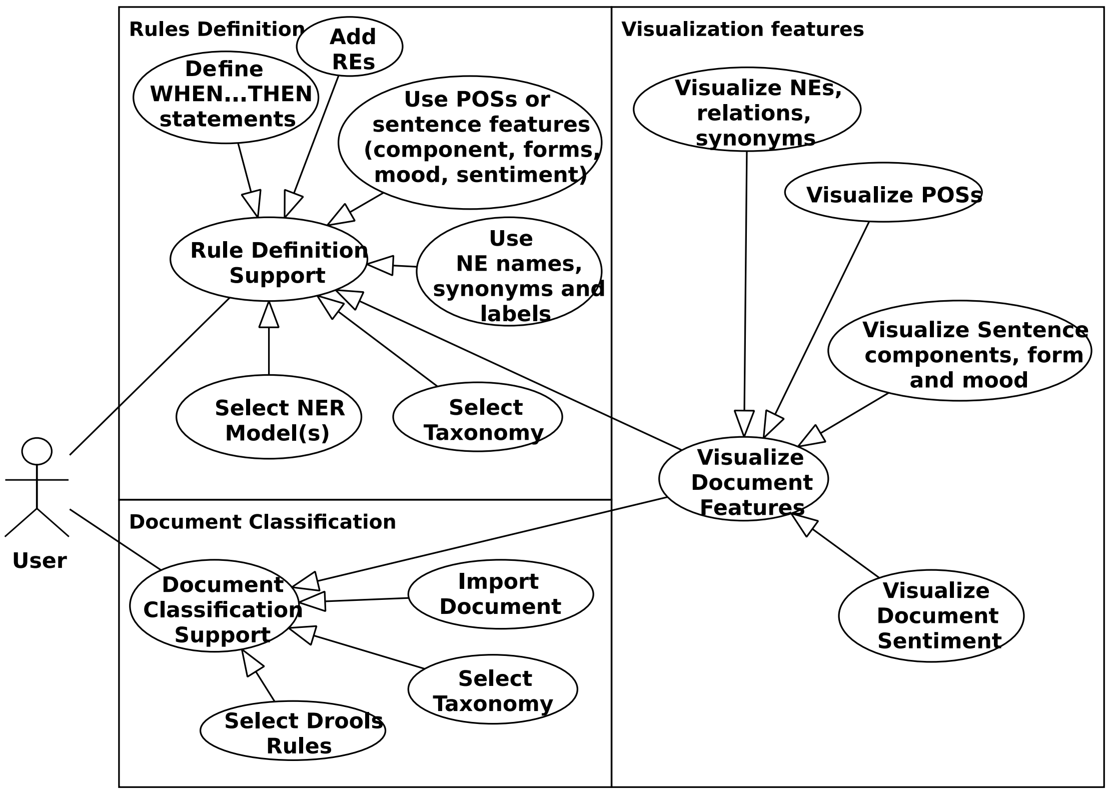
Figure 2
Unified modeling language use case diagram for NER


Corpus Annotation
Upload Domain Corpus
Annotate Domain Corpus
Select data cleanup options
Define NE Names, Synonyms Labels
Generate/ Upload Testing Data
Generate/ Upload Training Data
Corpus Annotation Support
User
Model Training
Model Training Support
Prepare Model
Test Model
Train Model
Add REs
Select word shape, stem, lemma
Select/customize CRF parameters
Select gazetteers, n-grams
Visualization features
Log Visualization Corpus Cleanup
Log Visualization Sentence Tokenization
Log Visualization Word Tokenization
Log Visualization POS Tokenization
Log visualization of executed NER preliminary steps
Log Visualization Token Annotation
Visualize Sentence features
Visualize NEs relations, synonyms
Visualize Standard Metrics P, R, F1
Visualize POS

definition—to execute all pipeline stages for training a NER model. For instance, users may upload a domain-specific data dump and perform data cleaning, followed by automatic annotation using Beginning, Inside, Outside (BIO) tagging format [22]. NE definitions include the original term (e.g., "JavaScript"), its synonyms (e.g., "js" and "Javascript"), and the corresponding domain-specific label. These annotations are used to generate training and testing sets. Given that supervised ML techniques demand extensive labeled datasets [21], this feature simplifies the process for domain experts while maintaining annotation quality. To prevent overfitting, users can revise annotations prior to training [75]. The system also supports CRF customization using the Stanford CoreNLP framework, allowing configuration of local/global features and the use of gazetteers. Users can also define REs to detect complex NEs. These configurations support flexible, iterative training and testing of NER models, even with minimal datasets. The training process is fully traceable through step-by-step logs. Once trained, model performance can be evaluated using Precision, Recall, and F1 scores. Additionally, users can visually inspect domain-specific NEs within sample documents, verify their correct labeling, and explore relationships with other NEs and POS tags. These visualizations are essential for building user trust in both the NER and DC subsystems, especially in the semantic rule formulation phase.

Our second use case diagram (Figure 3) focuses on DC. In this scenario, users can classify text documents within KM-EP by formulating handcrafted rules using our RBES approach. The system supports rule definition in a human-readable syntax, employing WHEN…ELSE statements to enhance usability and clarity. Users begin by selecting a relevant taxonomy within KM-EP to define the target classification categories for their documents. Rules can be constructed using domain-specific NEs —including their original names, synonyms, and associated labels —extracted from previously trained NER models. To link the rules with semantic context, users assign the appropriate domain NER models, ensuring alignment between classification logic and NER. In addition to NEs, users can incorporate a range of linguistic features, such as POS tags, sentence structure (including components, mood, and form), and document sentiment, allowing for more granular semantic analysis. REs may also be used to define complex rule conditions. The system enables automatic classification of imported documents. After documents and taxonomies are selected using KM-EP functionalities, the system applies predefined rules to assign taxonomy categories based on document content. Throughout this process, users are supported by rich visualization tools that display POS tags, NEs, their semantic relationships, and overall document sentiment. These visual aids help users understand a document's semantic structure

Figure 3
Unified modeling language use case diagram for DC


Rules Definition
Add REs
Define WHEN...THEN statements
Use POSs or sentence features (component, forms, mood, sentiment)
Rule Definition Support
Use NE names, synonyms and labels
Select NER Model(s)
Select Taxonomy
Visualization features
Visualize NEs, relations, synonyms
Visualize POSs
Visualize Sentence components, form and mood
Visualize Document Features
Visualize Document Sentiment
User
Document Classification
Document Classification Support
Import Document
Select Taxonomy
Select Drools Rules

and facilitate the identification of relevant features for rule construction and refinement.

3.3. Implementation of SNERC

This chapter outlines the prototypical implementation of SNERC, which provides comprehensive support for NER and DC within the KM-EP ecosystem. The implementation addresses all previously identified RC1–RC9 and is designed to offer users a flexible environment to build, customize, and evaluate ML-based NER models. In addition, SNERC integrates rule-based DC capabilities through an RBES, enhanced with standard NLP techniques, NER, and SPM, enabling effective semantic processing within KM-EP.

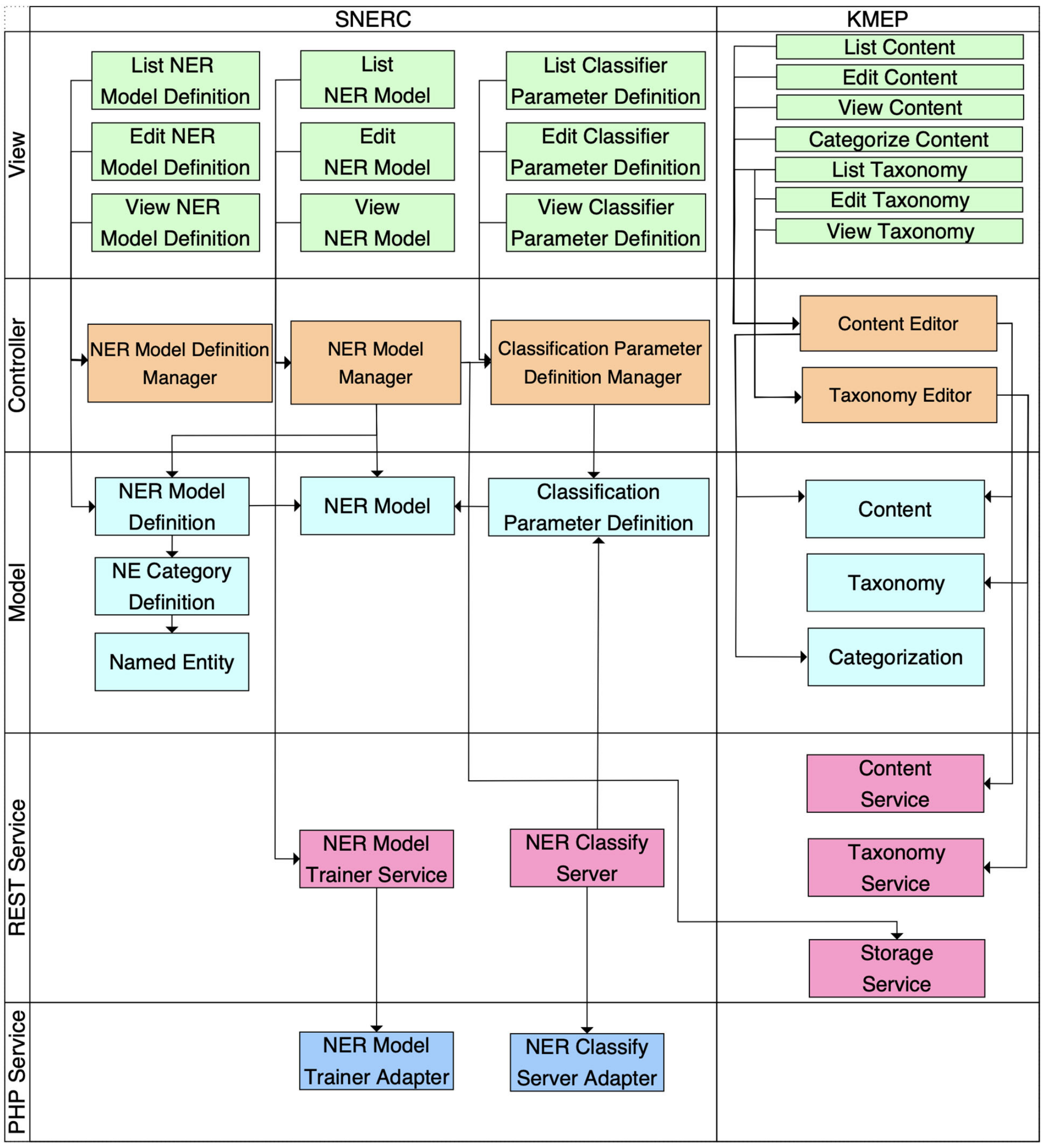
3.3.1. Development environment and tools

The SNERC prototype was developed using a hybrid technology stack that includes two RESTful services and multiple components designed to integrate seamlessly with the core infrastructure of KM-EP. The development team, consisting of two software engineers and one DevOps specialist collaborating remotely, employed modern software engineering tools and practices to ensure efficient development, testing, and deployment workflows. To enable a modular and isolated development environment, all SNERC components were containerized using Docker[1], which facilitated automated deployment and eliminated dependency conflicts across services. The project utilized GitLab[2] as a DevOps lifecycle platform, supporting source code versioning, continuous integration/ continuous deployment (CI/CD) pipelines, and native integration with Docker. For managing reproducible virtual environments, Vagrant[3] was used, allowing consistent development setups across the team. Dedicated Docker containers were created for each SNERC module. The KM-EP system itself runs in a PHP-based Docker container using the Symfony framework. Separate containers were set up for MariaDB, used for relational data storage, and Apache Solr, used for indexing and search functionalities required by KM-EP. The SNERC RESTful services —namely the NER Model Trainer Service and the NER Classify Server—were implemented in Java and deployed using an OpenJDK container image. All containers were obtained from Docker Hub, with detailed information on container setup and orchestration included as part of the system documentation.

The SNERC components were implemented using a combination of Java for backend RESTful services, PHP, Symfony, and AngularJS for frontend and integration within KM-EP, and MySQL for managing NER model definitions and classification parameters. This combination of technologies ensured compatibility with KM-EP's existing architecture while supporting scalable and efficient model development. SNERC derives its name from its foundational reliance on the Stanford CoreNLP framework, which serves as the backbone for training and executing ML-based NER models, as detailed in Section 2.2. The system architecture follows the Model-View-Controller (MVC) design pattern, ensuring clear separation of concerns between the data model, business logic, and user interface components. Figure 4 illustrates the complete architecture of SNERC, showing the integration of all components related to NER and DC within the KM-EP platform and highlighting how

[1]https://www.docker.com
[2]https://www.gitlab.com
[3]https://www.vagrantup.com

Figure 4
Integration architecture of SNERC and KM-EP



these components interact to support the system's end-to-end functionality.

3.3.2. SNERC components for NER

This section describes the implementation of the SNERC components supporting NER. Central to this functionality is the NER Model Definition Manager, which enables users to manage NER model definitions by configuring and customizing the preliminary steps required for model training. These steps include corpus upload, data cleanup, automatic annotation, dataset splitting, model training, and feature fine-tuning. The component leverages the Stanford CoreNLP API, which offers support for training and testing CRF-based NER models. Users can provide essential metadata, upload domain-specific corpora, define text-cleaning options, and annotate text using the BIO tagging format. Further configuration includes specifying domain-specific NEs, uploading gazetteers, defining CoreNLP-compatible REs, and customizing CRF parameters. Once configured, users may preview and initiate model training. The component is implemented using a split architecture: the frontend, built with AngularJS, collects user input and transmits it to the backend, implemented in Symfony, Java, and REST services. The backend delegates execution to the NER Model Trainer REST service, which conducts the model training and stores the results as JSON objects in the KM-EP database. The NER Model Manager component is responsible for maintaining trained NER models. It allows users to modify training and testing data, edit generated RE rules, retrain models, or remove outdated ones. It also supports uploading external models that have been trained using other CoreNLP-based systems. This component follows the same frontend-backend separation. The frontend provides a visual list of available models, along with key performance metrics—Precision, Recall, and $F1$-Score. The NER Model Trainer REST service encapsulates the entire model preparation and training pipeline.

Preparation involves automatic annotation of the uploaded corpus, splitting the annotated corpus into training and testing datasets, and generating REs for NE identification. The training phase then operates on the prepared data to build a CoreNLP-compliant NER model. Upon completion, the REST service returns a JSON object containing execution status, training logs, a Universally Unique Identifier, model and token URLs, generated REs, and evaluation metrics. The service exposes methods for preparing, training, updating, deleting, and uploading training and testing data sets.

3.3.3. SNERC components for DC

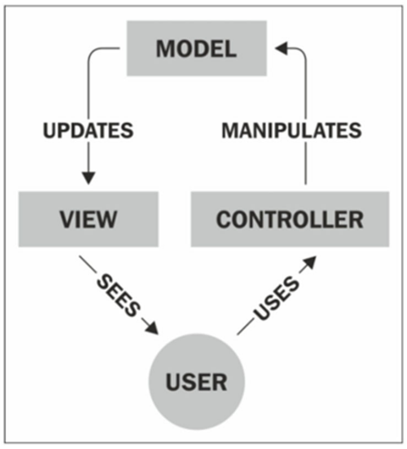
The SNERC architecture for DC consists of several core components. The "Classification Parameter Definition Manager" allows users to define semantic rules for classifying documents into taxonomies within KM-EP. It integrates with the "NER Model Manager" to connect trained NER models to these rules, enabling the use of domain-specific NEs, categories, and synonyms. This component is also linked to the KM-EP's "Content Editor and Taxonomy Editor," utilizing Simple Knowledge Organization System identifiers for taxonomy categories [77]. The NER Classify Server is an external REST service responsible for the automatic classification of documents into KM-EP taxonomies. Its architecture includes several functional modules. The Document module stores metadata and classification rules. The Linguistic Analyzer performs semantic analysis on documents, extracting NEs, NE categories, synonyms, POS tags, sentence structure (components, mood, and form), sentiment, and applying SPM. The NER Model module references trained models used in classification. Finally, the Drools Rules Helper applies Drools-based semantic rules on the server to execute DC.

3.3.4. System server specification and integration

To integrate SNERC seamlessly into KM-EP, the system's two REST services—NER Model Trainer and NER Classify Server—were implemented as external modules. This decoupled architecture ensures that NER and NLP features are accessible and maintainable without affecting KM-EP's core functionality. All communication between SNERC and KM-EP follows REST principles using the HTTP protocol. Two adapters were developed to facilitate communication. The NER Model Trainer Adapter invokes the REST service for model preparation and training, while the NER Classify Server Adapter retrieves domain-specific NE and NLP features to support rule construction in the DC process. Data exchange between KM-EP and the external services is encoded in JSON and represented internally using Plain Old Java Objects (POJOs) [78]. POJOs are simple Java classes with no framework dependencies and are used for structuring and transferring data. KM-EP is built on the PHP Symfony framework, which implements the MVC design pattern. In this architecture, the Model layer defines and manages application data; changes are propagated to the View layer, which renders the interface for the user. The Controller orchestrates application logic and data flow between Model and View.

The integration architecture of SNERC with KM-EP (illustrated in Figure 4) extends the MVC pattern with the following layers: the External Service layer, which hosts the two REST services for NER and DC; the Service layer, which includes the Symfony adapters that connect KM-EP with these services; the Model layer, which comprises Symfony classes responsible for persisting data and transferring it between systems; the Controller layer, which manages business logic for SNERC sub-modules; and the View layer, which presents NER model definitions and rule parameters in the frontend interface (as shown in Figure 5).

Figure 5
General life cycle of an MVC-based application



4. Experiment Results

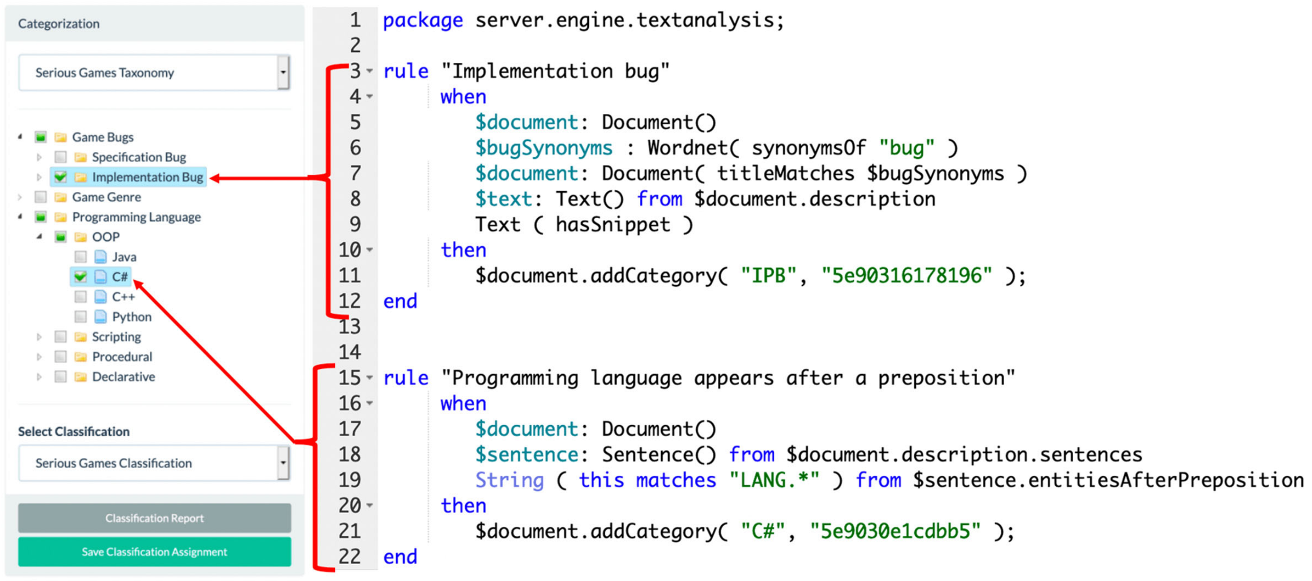
This section presents the experimental setup, proof of concept, and evaluation results for validating the SNERC prototype. The experiments demonstrate how discussions on Stack Overflow related to SG can be automatically classified within the RAGE platform using SNERC components. The evaluation focuses on the effectiveness of the system in supporting both NER and DC.

4.1. Proof of concept: classification workflow

The proof of concept begins with the use of an existing NER model integrated into the system. Through the Classification Parameter Definition Manager, users can define classification parameters that associate extracted entities from discussion texts with relevant taxonomies maintained in KM-EP.

To illustrate the process, consider a Stack Overflow post with the title "bug in my game loop," keywords such as "cocoa-touch" and "nstimer," and a description stating, "I am making a game on Xcode 5. I am using a nstimer in C# and there may be a bug in my game loop. Can you help me please. All help is great. <code>... </code>." Based on this content, and using the rule-based classification approach implemented with Drools, the document can be automatically classified under two taxonomies: Video Game Bug and Programming Language. The classification process starts by creating a Classification Parameter Definition, in which users select the appropriate taxonomy and associate it with trained NER models for entity extraction. Following this step, classification rules are authored using the WHEN... THEN syntax supported by the Drools rule engine. These rules refer to detected NEs, such as C# labeled as a programming language (LANG) and cocoa-touch labeled as a tool (TOOL), and apply additional techniques including linguistic analysis, web mining, and syntactic pattern matching. Figure 6 illustrates examples of such rules. In Rule 1, WordNet is used to detect the presence of the term "bug" or its synonyms within the post title (line 6–7), and a check is performed to determine whether the description contains a code snippet (line 8–9). If both conditions are met, the post is classified under the Implementation Bug category within the Video Game Bug taxonomy. Rule 2 evaluates whether a NE of type LANG, such as C#, appears after a preposition in the description. If this condition holds, the post is classified under the C# category in the Programming Language taxonomy. This workflow demonstrates how SNERC enables structured and

Figure 6
Selected categories and their rules



```
package server.engine.textanalysis;

rule "Implementation bug"
    when
        $document: Document()
        $bugSynonyms : Wordnet( synonymsOf "bug" )
        $document: Document( titleMatches $bugSynonyms )
        $text: Text() from $document.description
        Text ( hasSnippet )
    then
        $document.addCategory( "IPB", "5e90316178196" );
end


rule "Programming language appears after a preposition"
    when
        $document: Document()
        $sentence: Sentence() from $document.description.sentences
        String ( this matches "LANG.*" ) from $sentence.entitiesAfterPreposition
    then
        $document.addCategory( "C#", "5e9030e1cdbb5" );
end
```

reproducible DC by combining NE extraction with domain-specific taxonomy mapping.

To support testing and validation, an interactive user interface was developed. This interface allows users to input free text, apply the defined classification rules, and review the results through a detailed Classification Report. The report visualizes both annotation and classification outcomes and showcases various NLP features extracted using Stanford CoreNLP. These include recognized categories, sentence segmentation, part-of-speech tagging, NER, syntactic dependencies, and sentiment analysis. For greater transparency and flexibility, the complete CoreNLP output is also accessible in JSON format through the interface. The integration of classification, visualization, and NLP inspection capabilities within a single interface underscores the practical usability of the SNERC prototype for both research and operational purposes.

4.2. Qualitative evaluation: NER and annotation tasks

The qualitative evaluation assessed the ability of both beginners (newbies) and experts to perform critical NER tasks, namely data cleanup and data annotation. Participants were provided with tutorials, usage guidelines, and a domain-specific corpus consisting of Stack Overflow discussions related to programming languages. These texts included a range of NEs, such as official names, synonyms, and variations of programming languages. In Task 1.1, participants were asked to clean the corpus by removing irrelevant elements like HTML tags and URLs while preserving meaningful textual content. The results revealed that beginners achieved a higher Recall of 100% but slightly lower Precision (86.61%), whereas experts demonstrated perfect Precision (100%) but lower Recall (79.17%). Notably, some beginners outperformed experts, with one beginner achieving both 100% Precision and Recall. In Task 1.2, participants manually annotated tokens using the BIO tagging format. While both groups performed well, experts achieved a higher average *F*1-score of 95% compared to beginners, who reached an average *F*1-score of 85.77%. Some beginners struggled with the detailed annotation process, which slightly impacted their overall performance. These findings, summarized in Table 2, indicate that SNERC enables users with limited experience to achieve strong outcomes in key NER tasks, providing a robust foundation for training high-quality NER models.

Table 2
Precision, recall and F1 for Task 1.1 and Task 1.2

	Task 1.1			Task 1.2		
Participant	Precision (%)	Recall (%)	F1 (%)	Precision (%)	Recall (%)	F1 (%)
Newbie1	85.71	100.00	92.31	100.00	100.00	100.00
Newbie2	75.00	100.00	85.71	83.33	55.56	66.67
Newbie3	100.00	100.00	100.00	100.00	77.78	87.50
Newbie4	85.71	100.00	92.31	88.89	88.89	88.89
Expert1	100.00	83.33	90.91	72.73	88.89	80.00
Expert2	100.00	66.67	80.00	100.00	100.00	100.00
Expert3	100.00	83.33	90.91	100.00	100.00	100.00
Expert4	100.00	83.33	90.91	100.00	100.00	100.00

Table 3
Responses (%) for the usability questionnaire

Scale	This tool's capabilities meet my requirements. (%)	Using this tool is a frustrating experience. (%)	This tool is easy to use. (%)	I have to spend too much time correcting things with this tool. (%)
Strongly agree	12.5	0.0	0.0	0.0
Agree	25.0	0.0	25.0	0.0
Somewhat agree	25.0	0.0	50.0	25.0
Neither agree nor disagree	12.5	12.5	12.5	12.5
Somewhat disagree	12.5	12.5	0.0	50.0
Disagree	12.5	62.5	12.5	12.5
Strongly disagree	0.0	12.5	0.0	12.5

4.3. Quantitative evaluation: usability, usefulness, and interface assessment

To further evaluate the SNERC prototype, a quantitative walkthrough experiment was conducted involving participants with backgrounds in software engineering, ML, and data science. The experiment involved ten user accounts, including eight participants and two administrators. Participants performed tasks related to system configuration, NER model training, and DC, after which their feedback was collected through standardized questionnaires, including UMUX and USE, along with open-ended questions.

The usability evaluation, shown in Table 3, revealed that 62.5% of participants disagreed with the statement that using SNERC was frustrating, and 50% agreed that the tool was easy to use. Although some beginners required more time to familiarize themselves with the system interface, overall feedback was positive.

The usefulness evaluation, summarized in Table 4, demonstrated that most participants found SNERC helpful in enhancing productivity and supporting task completion. However, some beginners rated the system lower in aspects such as flexibility and customization, indicating areas for future improvement.

Regarding the user interface, feedback was overwhelmingly positive. As shown in Table 5, participants appreciated the speed of SNERC components, the clarity of the information presented, and the visual quality of the interface. In particular, 37.5% of participants strongly agreed that the components operated quickly, and a similar proportion praised the intuitive layout and clear presentation of explanations.

Evaluation of the NER feature set indicated that 37.5% of users found the NER Model Definition Manager effective for guiding model training, while 50% valued the Preview feature for verifying results before finalization. Although most participants expressed satisfaction with the available documentation and support, feedback also highlighted a desire for greater customization options and additional tutorial resources.

In summary, the experiments presented in this section demonstrate that SNERC provides an effective and practical solution for NER and DC in the context of Stack Overflow discussions related to SGs. Through the integration of NER models, rule-based classification with Drools, and domain-specific taxonomy mapping within KM-EP, SNERC enables structured, transparent, and reproducible classification processes. The developed interface supports intuitive user interaction, detailed visualization of NLP features, and transparent reporting of classification outcomes. Evaluation results confirmed that both beginners and experts could successfully complete essential NER tasks, with some beginners achieving outstanding Recall and Precision in data cleanup. Quantitative results further validated the system's usability and usefulness, while highlighting directions for future enhancements such as broader format support, additional NLP framework integration, and improved customization options. Overall, the SNERC prototype effectively addresses key challenges in NER and DC, demonstrating strong potential for broader application within knowledge management environments like RAGE and KM-EP.

Table 4
Responses (%) for the usefulness questionnaire

Scale	It helps me be more effective.	It helps me be more productive.	It is useful.	It gives me more control over the activities in my work.	It makes the things I want to accomplish easier to get done.	It saves me time when I use it.	It meets my needs.	It does everything I would expect it to do.
Strongly agree	12.5	12.5	37.5	0.0	25.0	37.5	12.5	25.0
Agree	37.5	25.0	50.0	25.0	0.0	12.5	12.5	37.5
Somewhat agree	12.5	25.0	0.0	25.0	62.5	37.5	37.5	25.0
Neither agree nor disagree	25.0	25.0	0.0	25.0	0.0	0.0	25.0	0.0
Somewhat disagree	0.0	0.0	0.0	12.5	0.0	0.0	0.0	0.0
Disagree	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
Strongly disagree	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5
Responses (%) for the user interface questionnaire

Scale	All SNERC components work fast	The user interface feels good	Buttons, images, and texts are in the right position	Enough information and explanations are presented	The images and icons look good
Strongly agree	37.5	12.5	25.0	0.0	37.5
Agree	37.5	62.5	37.5	37.5	37.5
Somewhat agree	12.5	25.0	12.5	62.5	25.0
Neither agree nor disagree	12.5	0.0	25.0	0.0	0.0
Somewhat disagree	0.0	0.0	0.0	0.0	0.0
Disagree	0.0	0.0	0.0	0.0	0.0
Strongly disagree	0.0	0.0	0.0	0.0	0.0

5. Conclusion and Future Work

The growing challenge of organizing and accessing relevant textual information, particularly within domain-specific communities, can be effectively addressed through the combined application of NER and DC. Although NER enables semantic enrichment of texts and DC supports structured categorization, both techniques often present usability barriers for users without programming or ML expertise. This study introduced SNERC, a system specifically designed to bridge this gap by providing an intuitive and effective environment for supporting NER and rule-based DC, catering to both novice and expert users alike. By integrating SNERC within a KMS such as KM-EP, we enable enhanced IR, faceted browsing, and taxonomy-based organization of unstructured discussion content. The work involved a conceptual and technical design of the system, leveraging Stanford CoreNLP for NLP feature extraction and Drools for flexible, rule-based DC. A comprehensive evaluation combining proof-of-concept demonstrations, qualitative user studies, and quantitative usability assessments confirmed that effective implementation of NER and DC relies on seamless integration with the technological infrastructure of the host platform. SNERC achieves this by enabling domain-specific NER model training, transparent rule authoring, and dynamic classification parameter management, thus facilitating scalable information extraction and document organization within KM-EP.

Looking forward, future work will focus on expanding SNERC's capabilities by incorporating support for multiple NLP frameworks, thereby enabling users to leverage and compare diverse NER approaches for improved domain adaptability. Enhancing the system's ability to manage various ML model formats will further strengthen interoperability and user workflow flexibility. To reduce entry barriers for non-technical users, future iterations should also integrate intelligent support tools, such as rule definition wizards with auto-completion features based on recognized entities and syntactic patterns. Finally, conducting large-scale evaluations with domain experts, particularly in fields like healthcare where information extraction from specialized texts such as Clinical Practice Guidelines is critical, will be essential to further validate SNERC's applicability and impact in real-world, high-stakes environments.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Philippe Tamla: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Florian Freund:** Methodology, Software, Validation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Matthias Hemmje:** Conceptualization, Validation, Supervision, Project administration, Funding acquisition.

References

[1] Jehangir, B., Radhakrishnan, S., & Agarwal, R. (2023). A survey on named entity recognition—Datasets, tools, and methodologies. *Natural Language Processing Journal*, *3*, 100017. https://doi.org/10.1016/j.nlp.2023.100017

[2] Wagh, V., Khandve, S., Joshi, I., Wani, A., Kale, G., & Joshi, R. (2021). Comparative study of long document classification. In *2021 IEEE Region 10 Conference*, 732–737. https://doi.org/10.1109/TENCON54134.2021.9707465

[3] Nawroth, C., Engel, F., & Hemmje, M. (2020). Emerging named entity recognition in a medical knowledge management ecosystem. In *International Conference on Knowledge Engineering and Ontology Development*, 29–41. https://doi.org/10.5220/0010061200290041

[4] Schmidt, R., Emmerich, K., & Schmidt, B. (2015). Applied games–in search of a new definition. In *Entertainment Computing: 14th International Conference*, 100–111. https://doi.org/10.1007/978-3-319-24589-8_8

[5] Michael, D. R., & Chen, S. L. (2005). *Serious games: Games that educate, train, and inform.* USA: Course Technology.

[6] Goli, A., Teymournia, F., Naemabadi, M., & Garmaroodi, A. A. (2022). Architectural design game: A serious game approach to promote teaching and learning using multimodal interfaces. *Education and Information Technologies*, *27*(8), 11467–11498. https://doi.org/10.1007/s10639-022-11062-z

[7] Hammady, R., & Arnab, S. (2022). Serious gaming for behaviour change: A systematic review. *Information*, *13*(3), 142. https://doi.org/10.3390/info13030142

[8] Vasudevamurt, V. B., & Uskov, A. (2015). Serious game engines: Analysis and applications. In *2015 IEEE International Conference on Electro/Information Technology*, 440–445. https://doi.org/10.1109/EIT.2015.7293381

[9] Ullah, M., Amin, S. U., Munsif, M., Yamin, M. M., Safaev, U., Khan, H., . . . , & Ullah, H. (2022). Serious games in science education: A systematic literature. *Virtual Reality & Intelligent Hardware*, *4*(3), 189–209. https://doi.org/10.1016/j.vrih.2022.02.001

[10] Barianos, A. K., Papadakis, A., & Vidakis, N. (2022). Content manager for serious games: Theoretical framework and digital platform. *Advances in Mobile Learning Educational Research*, *2*(1), 251–262. https://doi.org/10.25082/AMLER.2022.01.009

[11] Forschungsinstitut für Telekommunikation und Kooperation. (2025). *FTK e.V. Research Institute for Telecommunications and Cooperation*. Retrieved from: http://www.ftk.de

[12] Nunamaker, J. F., Chen, M., & Purdin, T. D. M. (1990). Systems development in information systems research. *Journal of Management Information Systems*, *7*(3), 89–106. https://doi.org/10.1080/07421222.1990.11517898

[13] Choi, H.-J., Ahn, J.-C., Jung, S.-H., & Kim, J.-H. (2020). Communities of practice and knowledge management systems: Effects on knowledge management activities and innovation performance. *Knowledge Management Research & Practice*, *18*(1), 53–68. https://doi.org/10.1080/14778238.2019.1598578

[14] Tamla, P., Freund, F., Hemmje, M., & Mc Kevitt, P. M. (2022). Evaluation of a system for named entity recognition in a knowledge management ecosystem. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, 2*, 19–31. https://doi.org/10.5220/0011374000003335

[15] Witeck, G. R., Alves, A. C., & Bernardo, M. H. (2021). Bloom taxonomy, serious games and lean learning: What do these topics have in common? In *Learning in the Digital Era 7th European Lean Educator Conference*, 308–316. https://doi.org/10.1007/978-3-030-92934-3_31

[16] Chiche, A., & Yitagesu, B. (2022). Part of speech tagging: A systematic review of deep learning and machine learning approaches. *Journal of Big Data*, *9*(1), 10. https://doi.org/10.1186/s40537-022-00561-y

[17] Geng, B. (2022). Text segmentation for patent claim simplification via bidirectional long-short term memory and conditional random field. *Computational Intelligence*, *38*(1), 205–215. https://doi.org/10.1111/coin.12455

[18] Patil, N., Patil, A., & Pawar, B. V. (2020). Named entity recognition using conditional random fields. *Procedia Computer Science*, *167*, 1181–1188. https://doi.org/10.1016/j.procs.2020.03.431

[19] Shelar, H., Kaur, G., Heda, N., & Agrawal, P. (2020). Named entity recognition approaches and their comparison for custom NER model. *Science & Technology Libraries*, *39*(3), 324–337. https://doi.org/10.1080/0194262X.2020.1759479

[20] Benajiba, Y., Diab, M., & Rosso, P. (2009). Using language independent and language specific features to enhance Arabic named entity recognition. *The International Arab Journal of Information Technology*, *6*(5), 464–473.

[21] Nasar, Z., Jaffry, S. W., & Malik, M. K. (2021). Named entity recognition and relation extraction: State-of-the-art. *ACM Computing Surveys*, *54*(1), 20. https://doi.org/10.1145/3445965

[22] Alshammari, N., & Alanazi, S. (2021). The impact of using different annotation schemes on named entity recognition. *Egyptian Informatics Journal*, *22*(3), 295–302. https://doi.org/10.1016/j.eij.2020.10.004

[23] Ranjan, N. M., & Prasad, R. S. (2023). A brief survey of text document classification algorithms and processes. *Journal of Data Mining and Management*, *8*(1), 6–11. https://doi.org/10.46610/JoDMM.2023.v08i01.002

[24] Sreekumar, A., Reshma, R., & Athira, B. (2023). Comparative study of deep learning models for document classification. In *2023 9th International Conference on Smart Computing and Communications*, 512–517. https://doi.org/10.1109/ICSCC59169.2023.10335027

[25] Hase, V. (2022). Automated content analysis. In F. Oehmer-Pedrazzi, S. H. Kessler, E. Humprecht, K. Sommer, & L. Castro (Eds.), *Standardisierte Inhaltsanalyse in der Kommunikationswissenschaft – Standardized content analysis in communication research: Ein Handbuch-A handbook* (pp. 23–36). Springer VS. https://doi.org/10.1007/978-3-658-36179-2_3

[26] Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, *1*(4), 309–317. https://doi.org/10.1147/rd.14.0309

[27] Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47. https://doi.org/10.1145/505282.505283

[28] Mohammed, S. M., Jacksi, K., & Zeebaree, S. R. M. (2021). A state-of-the-art survey on semantic similarity for document clustering using GloVe and density-based algorithms. *Indonesian Journal of Electrical Engineering and Computer Science*, *22*(1), 552–562. http://doi.org/10.11591/ijeecs.v22.i1.pp552-562

[29] Thielmann, A., Weisser, C., Krenz, A., & Säfken, B. (2023). Unsupervised document classification integrating web scraping, one-class SVM and LDA topic modelling. *Journal of Applied Statistics*, *50*(3), 574–591. https://doi.org/10.1080/02664763.2021.1919063

[30] Kim, Y. S., Park, S. S., Deards, E., & Kang, B. H. (2004). Adaptive web document classification with MCRDR. In *International Conference on Information Technology: Coding and Computing, 1*, 476–480. https://doi.org/10.1109/ITCC.2004.1286502

[31] Olsson, J. S., Oard, D. W., & Hajič, J. (2005). Cross-language text classification. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 645–646. https://doi.org/10.1145/1076034.1076170

[32] Shen, D., Wu, J., Cao, B., Sun, J.-T., Yang, Q., Chen, Z., & Li, Y. (2009). Exploiting term relationship to boost text classification. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 1637–1640. https://doi.org/10.1145/1645953.1646192

[33] Biswas, P., Sharan, A., & Kumar, R. (2014). Question classification using syntactic and rule based approach. In *2014 International Conference on Advances in Computing, Communications and Informatics*, 1033–1038. https://doi.org/10.1109/ICACCI.2014.6968434

[34] Villena-Román, J., Collada-Pérez, S., Lana-Serrano, S., & González-Cristóbal, J. C. (2011). Hybrid approach combining machine learning and a rule-based expert system for text categorization. In *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, 323–328.

[35] Maynard, D., Bontcheva, K., & Augenstein, I. (2017). *Natural language processing for the semantic web*. Switzerland: Springer. https://doi.org/10.1007/978-3-031-79474-2

[36] Liu, M., Peng, X., Jiang, Q., Marcus, A., Yang, J., & Zhao, W. (2018). Searching stackoverflow questions with multi-faceted categorization. In *Proceedings of the 10th Asia-Pacific Symposium on Internetware*, 10. https://doi.org/10.1145/3275219.3275227

[37] Purohit, A., & Patheja, P. S. (2023). Product review opinion based on sentiment analysis. *Journal of Intelligent & Fuzzy Systems:*

Applications in Engineering and Technology, *44*(2), 3153–3169. https://doi.org/10.3233/JIFS-213296

[38] Serafini, F., & Reid, S. F. (2023). Multimodal content analysis: Expanding analytical approaches to content analysis. *Visual Communication*, *22*(4), 623–649. https://doi.org/10.1177/1470357219864133

[39] Talukdar, J., Singh, T. P., & Barman, B. (2023). *Artificial intelligence in healthcare industry*. Singapore: Springer. https://doi.org/10.1007/978-981-99-3157-6

[40] Abueleiwa, M. H. S., Harara, F. E. S., Al-Ghoul, M. M. K., Okasha, S. M., & Abu-Naser, S. S. (2022). Rule based system for diagnosing bean diseases and treatment. *International Journal of Engineering and Information Systems*, *6*(5), 67–74.

[41] Dandea, V., & Grigoras, G. (2023). Expert system integrating rule-based reasoning to voltage control in photovoltaic-systems-rich low voltage electric distribution networks: A review and results of a case study. *Applied Sciences*, *13*(10), 6158. https://doi.org/10.3390/app13106158

[42] Shishehchi, S., & Banihashem, S. Y. (2021). A rule based expert system based on ontology for diagnosis of ITP disease. *Smart Health*, *21*, 100192. https://doi.org/10.1016/j.smhl.2021.100192

[43] Reddy, B., & Fields, R. (2022). From past to present: A comprehensive technical review of rule-based expert systems from 1980–2021. In *Proceedings of the 2022 ACM Southeast Conference*, 167–172. https://doi.org/10.1145/3476883.3520211

[44] Negnevitsky, M. (2005). *Artificial intelligence: A guide to intelligent systems* (2nd ed.). USA: Addison-Wesley.

[45] Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., & Lederberg, J. (1993). DENDRAL: A case study of the first expert system for scientific hypothesis formation. *Artificial Intelligence*, *61*(2), 209–261. https://doi.org/10.1016/0004-3702(93)90068-M

[46] Swartout, W. R. (1985). Rule-based expert systems: The MYCIN experiments of the stanford heuristic programming project. *Artificial Intelligence, 26*(3), 364–366. https://doi.org/10.1016/0004-3702(85)90067-0

[47] Soe, S. M. M., & Zaw, M. P. P. (2008). Design and implementation of rule-based expert system for fault management. *World Academy of Science, Engineering and Technology*, *2*(12), 4022–4027. https://doi.org/10.5281/zenodo.1083855

[48] Julianti, M. R., Nurmaesah, N., & Prayogo, W. (2022). Expert system for diagnosing early symptoms of COVID-19 using the certainty factor method. *Journal Sisfotek Gloval*, *12*(1), 24–31. https://doi.org/10.38101/sisfotek.v12i1.475

[49] Nagaraj, P., & Deepalakshmi, P. (2022). An intelligent fuzzy inference rule-based expert recommendation system for predictive diabetes diagnosis. *International Journal of Imaging Systems and Technology*, *32*(4), 1373–1396. https://doi.org/10.1002/ima.22710

[50] van Melle, W. (1978). MYCIN: A knowledge-based consultation program for infectious disease diagnosis. *International Journal of Man-Machine Studies*, *10*(3), 313–322. https://doi.org/10.1016/S0020-7373(78)80049-2

[51] Alkahlout, M. A., Alsaqqa, A. H., Abu-Jamie, T. N., & Abu-Naser, S. S. (2021). An expert system diagnosing facial-swelling using CLIPS. *International Journal of Academic Information Systems Research*, *5*(5), 29–36.

[52] Friedman-Hill, E. J. (1997). *Jess, the Java expert system shell* (No. SAND-98-8206). Sandia National Laboratories. https://www.osti.gov/servlets/purl/565603

[53] Alnattah, A., Jajroudi, M., & Eslami, S. (2024). *Exploring rules engines as an artificial intelligence application in medicine: A scoping review*. Research Square. https://doi.org/10.21203/rs.3.rs-3839314/v1

[54] Ratan, R. A., & Ritterfeld, U. (2009). Classifying serious games. In U. Ritterfeld, M. Cody, & P. Vorderer (Eds.), *Serious games: Mechanisms and effects* (pp. 10–24). Routledge. https://doi.org/10.4324/9780203891650-10

[55] Dankov, Y., & Bontchev, B. (2020). Towards a taxonomy of instruments for facilitated design and evaluation of video games for education. In *Proceedings of the 21st International Conference on Computer Systems and Technologies*, 285–292. https://doi.org/10.1145/3407982.3408010

[56] van Roy, P. (2009). Programming paradigms for dummies: What every programmer should know. In G. Assayag & A. Gerzso (Eds.), *New computational paradigms for computer music* (pp. 9–47). Delatour France.

[57] Toftedahl, M., & Engström, H. (2019). A taxonomy of game engines and the tools that drive the industry. In *Proceedings of DiGRA 2019 Conference: Game, Play and the Emerging Ludo-Mix*, 1–17.

[58] Melo, F. S., Mascarenhas, S., & Paiva, A. (2018). A tutorial on machine learning for interactive pedagogical systems. *International Journal of Serious Games*, *5*(3), 79–112. https://doi.org/10.17083/ijsg.v5i3.256

[59] Truelove, A., de Almeida, E. S., & Ahmed, I. (2021). We'll fix it in post: What do bug fixes in video game update notes tell us? In *2021 IEEE/ACM 43rd International Conference on Software Engineering*, 736–747. https://doi.org/10.1109/ICSE43902.2021.00073

[60] Varvaressos, S., Lavoie, K., Gaboury, S., & Hallé, S. (2017). Automated bug finding in video games: A case study for runtime monitoring. *Computers in Entertainment*, *15*(1), 1. https://doi.org/10.1145/2700529

[61] Emond, B. (2021). Formal methods in human-computer interaction and adaptive instructional systems. In *Adaptive Instructional Systems. Design and Evaluation: Third International Conference*, 183–198. https://doi.org/10.1007/978-3-030-77857-6_12

[62] Mahatody, T., Sagar, M., & Kolski, C. (2010). State of the art on the cognitive walkthrough method, its variants and evolutions. *International Journal of Human–Computer Interaction*, *26*(8), 741–785. https://doi.org/10.1080/10447311003781409

[63] Nielsen, J. (1994). Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems*, 413–414. https://doi.org/10.1145/259963.260531

[64] Tümen Akyıldız, S., & Ahmed, K. H. (2021). An overview of qualitative research and focus group discussion. *International Journal of Academic Research in Education*, *7*(1), 1–15. https://doi.org/10.17985/ijare.866762

[65] Noushad, B., van Gerven, P. W. M., & de Bruin, A. B. H. (2024). Twelve tips for applying the think-aloud method to capture cognitive processes. *Medical Teacher*, *46*(7), 892–897. https://doi.org/10.1080/0142159X.2023.2289847

[66] Yacouby, R., & Axman, D. (2020). Probabilistic extension of precision, recall, and F1 score for more thorough evaluation of classification models. In *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*, 79–91. https://doi.org/10.18653/v1/2020.eval4nlp-1.9

[67] Ramachandran, S., Nor, R. N. H., & Jusoh, Y. Y. (2019). Usability assessment for the enhancement of quality of a

web portal interface. *International Journal of Engineering and Advanced Technology*, *9*(1), 1636–1640. https://doi.org/10.35940/ijeat.A2635.109119

[68] Ntoa, S. (2025). Usability and user experience evaluation in intelligent environments: A review and reappraisal. *International Journal of Human–Computer Interaction*, *41*(5), 2829–2858. https://doi.org/10.1080/10447318.2024.2394724

[69] Moulaei, K., Moulaei, R., & Bahaadinbeigy, K. (2024). The most used questionnaires for evaluating the usability of robots and smart wearables: A scoping review. *Digital Health*, *10*, 20552076241237384. https://doi.org/10.1177/20552076241237384

[70] Thielsch, M. T., & Stegemöller, I. (2010). *Münsteraner fragebogen zur evaluation-zusatzmodul diskussion (MFE-ZDi) [Münster questionnaire for the evaluation supplementary module discussion (MFE-ZDi)]*. Zusammenstellung Sozialwissenschaftlicher Items und Skalen. https://doi.org/10.6102/zis98

[71] Wallach, D., & Scholz, S. C. (2012). User-centered design: Why and how to put users first in software development. In A. Maedche, A. Botzenhardt, & L. Neer (Eds.), *Software for people: Fundamentals, trends and best practices* (pp. 11–38). Springer. https://doi.org/10.1007/978-3-642-31371-4_2

[72] Norman, D. A., & Draper, S. W. (1988). User centered system design: New perspectives on human-computer interaction. *The American Journal of Psychology, 101*(1), 148–151. https://doi.org/10.2307/1422802

[73] Vredenburg, K., Mao, J.-Y., Smith, P. W., & Carey, T. (2002). A survey of user-centered design practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 471–478. https://doi.org/10.1145/503376.503460

[74] Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The unified modeling language reference manual*. USA: Addison-Wesley.

[75] Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. USA: O'Reilly Media.

[76] Aubaid, A. M., & Mishra, A. (2020). A rule-based approach to embedding techniques for text document classification. *Applied Sciences*, *10*(11), 4009. https://doi.org/10.3390/app10114009

[77] Smith, A. (2022). Simple knowledge organization system (SKOS). *Ko Knowledge Organization*, *49*(5), 371–384. https://doi.org/10.5771/0943-7444-2022-5-371

[78] Rojas, J., Fragoso, O., Santaolaya, R., & Soto, J. (2020). Generation of POJOs and DAOs classes from metadata database. *IEEE Latin America Transactions*, *18*(9), 1547–1554. https://doi.org/10.1109/TLA.2020.9381796

How to Cite: Tamla, P., Freund, F., & Hemmje, M. (2025). SNERC: Enhancing Knowledge Management with Named Entity Recognition and Document Classification for Apply Gaming. *Artificial Intelligence and Applications*, *3*(4), 392–407. https://doi.org/10.47852/bonviewAIA52023841