American Journal of Advanced Technology and Engineering Solutions

Volume: 6; **Issue:** 1
Pages: 690-716
eISSN: 3067-5146

A Runtime Activity-Aware Framework for Dynamic Floating License Reclamation in Enterprise CAD Environments Study in Smart Manufacturing

Md Syeedur Rahman[1]; Sharmin Rahman[2];

[1]. Solution Developer, Plant Design I/O, Gonzales, Louisiana, USA; Email: syeed_rahman@live.com; https://orcid.org/0000-0001-8475-0615

[2]. Department of Electrical & Computer Engineering, Lamar University, Beaumont, Texas, USA; Email : sharmin.rahman89@outlook.com https://orcid.org/0009-0007-3444-9040

Doi: 10.63125/adwrax90

Received: 04 October 2025; **Revised:** 22 November 2025; **Accepted:** 05 December 2025; **Published:** 09 January 2026;

Abstract

Floating licensing has become the predominant software licensing model for enterprise Computer-Aided Design (CAD) applications, enabling organizations to optimize the utilization of expensive engineering software through concurrent license sharing. However, conventional floating license management mechanisms allocate licenses solely based on application execution and are unable to distinguish between actively used and idle application instances. Consequently, engineering users frequently retain valuable licenses while applications remain inactive during meetings, breaks, or prolonged periods of inactivity, leading to reduced license availability, increased user waiting time, and unnecessary software procurement costs. This paper proposes a Runtime Activity-Aware Framework for Dynamic Floating License Reclamation that enhances license utilization through continuous client-side monitoring of application runtime activity. The proposed framework periodically evaluates process activity using accumulated processor execution time over a configurable observation interval to determine whether a CAD application is actively performing computational work. When sustained inactivity is detected, the framework transparently suspends the application process while preserving its execution state. The suspension interrupts the application's communication with the floating license server, enabling automatic license reclamation without modifying either the CAD application or the license management infrastructure. When the user resumes work, the suspended application is restored seamlessly and automatically reacquires an available license, allowing uninterrupted continuation of the engineering session. The proposed framework is implemented as a lightweight client-side monitoring agent and validated through deployment in a real industrial engineering environment using a commercially deployed floating-license CAD platform. The framework was evaluated using longitudinal operational data collected from a production enterprise deployment, including engineering workload records and concurrent floating-license utilization data. The results indicate that runtime activity-aware license reclamation can improve enterprise software license utilization while preserving engineering workflow continuity through transparent application suspension and recovery with minimal computational overhead. The proposed framework provides a practical and vendor-independent approach for optimizing floating-license utilization in engineering organizations without requiring modifications to existing CAD applications or enterprise license servers. The study contributes a reusable methodology for intelligent software resource management that can be extended to a wide range of floating-license engineering applications.

Keywords

Floating License Management; Computer-Aided Design (CAD); Runtime Activity Monitoring; Software Asset Management, Process Suspension, Enterprise Engineering, License Reclamation, Resource Optimization

INTRODUCTION

Enterprise computer-aided design (CAD) systems have become indispensable components of modern engineering organizations, supporting product design, plant engineering, structural modeling, equipment layout, and digital asset development across diverse industrial sectors. As engineering projects continue to increase in complexity, organizations have become increasingly dependent on commercial CAD platforms that provide advanced modeling, visualization, simulation, and collaboration capabilities. Unlike many conventional desktop applications, professional engineering software is commonly distributed under concurrent (floating) licensing models, whereby a centralized license server dynamically allocates a limited pool of software licenses to users when applications are launched. Compared with individually assigned licenses, floating licensing enables multiple engineers to share software resources according to operational demand, thereby improving utilization efficiency while reducing software acquisition costs (Doloca & Ţănculescu, 2011; Machal-Fulks & Barnett, 2012).

Commercial CAD applications represent a substantial long-term investment for engineering organizations, particularly in industries such as oil and gas, manufacturing, power generation, shipbuilding, and infrastructure engineering. Consequently, organizations seek to maximize the utilization of available software licenses while minimizing unnecessary procurement of additional licenses. Comparative studies of commercial and open-source CAD platforms demonstrate that although open-source alternatives have matured considerably, commercial CAD software continues to dominate industrial engineering practice because of its advanced functionality, interoperability, and integration with enterprise Product Lifecycle Management (PLM) environments (Papachristou et al., 2019). As a result, floating licenses have become valuable organizational resources whose efficient allocation directly influences engineering productivity and operational performance.

In a typical floating-license architecture, a client application requests authorization from a centralized license server before execution begins. Once a license is granted, it remains reserved while the application maintains periodic communication with the license server. This architecture enables organizations to share a finite number of licenses among a larger engineering workforce and has become the preferred licensing model for many commercial engineering software vendors because it balances software availability with licensing costs (Doloca & Ţănculescu, 2011; Wikberg, 2010). Nevertheless, the same mechanism introduces a significant operational challenge. Licenses remain associated with running application sessions regardless of whether engineers are actively performing design activities. Consequently, software licenses may remain unavailable to other users even when the associated application is effectively idle.

This phenomenon creates an important paradox in enterprise engineering environments. During periods of high demand, engineers frequently encounter situations in which all floating licenses appear to be occupied, preventing new CAD sessions from being initiated. At the same time, a proportion of these allocated licenses may correspond to application instances that have been left open during meetings, documentation activities, design reviews, or other non-modeling tasks. The resulting mismatch between allocated licenses and actual engineering activity reduces the effective capacity of the license pool and may unnecessarily delay engineering work. Organizations often respond by purchasing additional software licenses to alleviate these shortages; however, the underlying problem may stem from inefficient utilization rather than insufficient licensing capacity. Given the substantial financial investment associated with commercial engineering software, improving license utilization represents both an operational optimization problem and a significant business opportunity (Machal-Fulks & Barnett, 2012; Doloca & Ţănculescu, 2011).

The broader discipline of Software Asset Management (SAM) has long recognized the importance of maximizing the value of software investments through systematic management of software assets, licensing agreements, compliance verification, and lifecycle governance (Phillips, 2017; David, 2023). Modern SAM practices seek to provide organizations with visibility into software deployment, usage, procurement, and licensing obligations while reducing legal and financial risks associated with non-compliance. Recent developments further explore the application of artificial intelligence to improve software asset management by supporting automated discovery, license analysis, and compliance assessment (Schlauch, 2023). Although these approaches significantly improve administrative control

over enterprise software portfolios, they primarily operate at organizational or management levels and provide limited support for making runtime decisions based on the operational state of individual engineering applications.

Similarly, research on enterprise software licensing has largely focused on contractual obligations, compliance management, software inventory, and governance frameworks rather than on optimizing license utilization during software execution (Machal-Fulks & Barnett, 2012; Alspaugh et al., 2010). Recent work on automated software license compliance has further demonstrated the importance of detecting licensing violations and ensuring that software usage remains consistent with licensing agreements (Wintersgill et al., 2025). While these studies contribute substantially to improving software governance and reducing licensing risks, they generally assume that an active software session represents legitimate resource utilization. Consequently, existing approaches provide limited mechanisms for distinguishing between actively used applications and application sessions that continue occupying floating licenses despite prolonged user inactivity.

The limitations of existing license management approaches become particularly evident in enterprise CAD environments, where application execution does not necessarily correspond to productive engineering work. Engineers routinely alternate between three-dimensional modeling, design reviews, documentation, meetings, simulations, and collaborative discussions while keeping CAD applications open throughout the working day. As a result, a running process cannot reliably indicate whether meaningful engineering activity is actually taking place. Conventional monitoring systems typically collect processor utilization, memory consumption, process status, and other infrastructure metrics to evaluate system health and application performance. Although these monitoring techniques are highly effective for operational management, they provide limited insight into the behavioral state of engineering applications and therefore cannot reliably determine when a floating license is genuinely required by an active user.

Recent advances in runtime software monitoring have demonstrated the importance of observing application behavior during execution to improve system performance, reliability, anomaly detection, and operational decision-making (Hasselbring & van Hoorn, 2020; Gaol et al., 2022). Modern monitoring frameworks continuously collect execution metrics such as processor activity, memory utilization, process status, thread behavior, and application events to support infrastructure management and performance optimization. These techniques have proven effective for identifying resource bottlenecks, diagnosing software failures, and improving service availability. However, their primary objective is to evaluate application or system health rather than to determine whether an application instance is actively supporting productive user work. Consequently, runtime monitoring solutions generally provide limited capability for interpreting user activity in the context of software license utilization.

The distinction between application execution and meaningful user activity is particularly significant in enterprise CAD environments. Unlike many business applications that execute short-lived transactions, engineering software often remains open for extended periods while engineers alternate between modeling, documentation, engineering calculations, design discussions, project meetings, and other collaborative activities. During these transitions, a CAD application may continue executing while exhibiting minimal computational activity or user interaction. Conversely, legitimate engineering operations such as model regeneration, rendering, or complex computational analyses may temporarily reduce user interaction despite representing active engineering tasks. These characteristics demonstrate that effective license management requires a more nuanced interpretation of runtime application behavior than can be achieved through simple process existence, instantaneous processor utilization, or application availability alone.

Behavior-aware software analysis has received increasing attention in software engineering research because runtime execution characteristics frequently provide information that cannot be inferred from static system observations. Studies investigating behavioral changes in software systems have shown that runtime execution patterns can reveal significant differences in application behavior that remain invisible during static analysis (Danglot et al., 2020). Likewise, research on software license compliance demonstrates that observing software behavior can improve the detection of abnormal or unauthorized

software usage beyond traditional inventory-based approaches (Wintersgill et al., 2025). Although these studies address different application domains, they collectively suggest that runtime behavioral information provides a richer foundation for operational decision-making than static resource observations alone. Nevertheless, these approaches primarily focus on compliance verification, anomaly detection, or software quality assurance rather than improving the operational utilization of floating software licenses.

The broader literature on resource optimization similarly emphasizes the importance of adaptive decision-making in environments where valuable resources are shared among multiple users. Research in software asset management consistently argues that effective utilization of software resources requires continuous visibility into software consumption patterns rather than periodic administrative auditing (Phillips, 2017; David, 2023). Likewise, enterprise software licensing research recognizes that organizations must balance software availability, licensing costs, and compliance obligations to maximize the return on software investments (Machal-Fulks & Barnett, 2012; Alspaugh et al., 2010). Despite these advances, existing studies primarily concentrate on organizational governance, procurement strategies, compliance assessment, and contractual management. Comparatively little attention has been devoted to dynamically reclaiming floating software licenses during application execution while preserving user sessions and minimizing workflow disruption.

This limitation is particularly relevant in enterprise engineering environments, where software availability directly influences engineering productivity and project delivery. Conventional approaches generally rely on administrative policies such as predefined inactivity timeouts, manual application closure, or license server configurations to recover unused licenses. While these mechanisms can reduce prolonged license retention, they often lack sufficient awareness of actual runtime application behavior and may either reclaim licenses prematurely or allow inactive sessions to retain valuable software resources for extended periods. Consequently, organizations continue to experience avoidable license shortages despite possessing sufficient licensed capacity under typical operating conditions.

Collectively, the existing literature demonstrates substantial progress in software asset management, enterprise software licensing, runtime monitoring, behavioral analysis, and software compliance. However, these research streams have evolved largely independently. Current software asset management solutions emphasize organizational governance but provide limited runtime awareness. Runtime monitoring systems effectively observe application execution but do not interpret activity from the perspective of floating license optimization. License compliance research focuses on contractual correctness rather than operational resource utilization. As a result, there remains a significant research gap concerning how runtime application activity can be translated into reliable, automated decisions for reclaiming floating software licenses without compromising user productivity or application continuity.

To address this gap, this paper proposes a Runtime Activity-Aware Framework for Dynamic Floating License Reclamation in Enterprise CAD Environments. Unlike conventional administrative or server-centric approaches, the proposed framework continuously evaluates the runtime activity of individual CAD applications at the client side to identify application sessions that are functionally inactive while preserving their execution state. Rather than terminating inactive applications or relying solely on predefined timeout policies, the framework temporarily suspends inactive application execution, allowing the associated floating license to be automatically released by the license server while maintaining the user's working environment. When engineering work resumes, the application is restored, enabling the license to be reacquired transparently with minimal interruption to the user.

The proposed approach offers several practical advantages for enterprise engineering organizations. First, it improves the availability of floating licenses by reducing unnecessary license occupation caused by prolonged inactive application sessions. Second, it increases the effective utilization of existing software assets, thereby reducing the need for additional license procurement and improving the return on software investment. Third, because application state is preserved throughout the suspension period, engineers can resume their work without repeating previous operations or reconstructing their design environment. Finally, the framework operates without requiring modifications to commercial CAD software or license server infrastructure, facilitating deployment within existing enterprise

engineering environments.

The principal contributions of this research are summarized as follows:

a. A runtime activity-aware monitoring framework that continuously evaluates the execution behavior of enterprise CAD applications to distinguish active engineering work from functionally inactive application sessions.
b. A dynamic floating license reclamation mechanism that temporarily suspends inactive application execution, enabling floating licenses to be automatically returned to the centralized license pool while preserving application state.
c. An industrial implementation and deployment demonstrating the practical feasibility of integrating the proposed framework within an operational enterprise engineering environment without modifying existing commercial CAD software or licensing infrastructure.
d. A comprehensive experimental evaluation assessing the effectiveness of the proposed framework in improving floating license availability, software utilization, and operational efficiency while maintaining minimal performance overhead and negligible disruption to engineering workflows.

The remainder of this paper is organized as follows. Section 2 reviews related work on software asset management, floating software licensing, runtime monitoring, and enterprise software resource optimization. Section 3 presents the architecture of the proposed runtime activity-aware license reclamation framework. Section 4 describes the activity monitoring and license reclamation methodology. Section 5 discusses the implementation details and industrial deployment. Section 6 presents the experimental evaluation and performance analysis. Finally, Sections 7 and 8 discuss the implications of the proposed approach, acknowledge its limitations, and summarize the principal conclusions of this study.

LITERATURE REVIEW

2.1 ENTERPRISE CAD SOFTWARE AND FLOATING LICENSE MANAGEMENT

Commercial Computer-Aided Design (CAD) systems constitute the technological foundation of many engineering organizations, supporting product development, plant engineering, structural design, and lifecycle documentation. Unlike conventional desktop software, enterprise CAD applications are typically licensed using concurrent (floating) licensing models, allowing a finite pool of software licenses to be dynamically shared among multiple users. This licensing strategy significantly reduces software acquisition costs while maintaining operational flexibility because organizations can provision licenses based on expected concurrent demand rather than total workforce size (Doloca & Țănculescu, 2011; Machal-Fulks & Barnett, 2012). Consequently, floating licensing has become the preferred deployment model for many high-value engineering applications.

Despite these advantages, efficient management of floating licenses remains challenging. Enterprise software licensing research has primarily focused on contractual obligations, entitlement management, procurement strategies, and compliance with licensing agreements (Machal-Fulks & Barnett, 2012; Wikberg, 2010). Existing studies emphasize administrative governance of software assets, including license inventories, entitlement reconciliation, and vendor compliance. While these approaches are essential for reducing legal and financial risks, they provide limited guidance for improving license utilization during software execution. In particular, they rarely consider situations in which application sessions remain active despite little or no productive user activity, leaving valuable licenses unavailable to other engineers.

The operational implications of inefficient license utilization are particularly significant in engineering organizations because commercial CAD software often represents one of the largest recurring software investments. Comparative analyses of CAD technologies continue to demonstrate the importance of professional commercial systems for industrial design and engineering despite the availability of open-source alternatives (Papachristou et al., 2019; Camba et al., 2016). Consequently, maximizing the utilization of existing license pools has become both an economic and operational objective for organizations seeking to improve engineering productivity without increasing software procurement costs.

2.2 SOFTWARE ASSET MANAGEMENT AND SOFTWARE LICENSE GOVERNANCE

Software Asset Management (SAM) has evolved into a mature discipline that enables organizations to manage software assets throughout their lifecycle, encompassing acquisition, deployment, maintenance, compliance, and retirement (Phillips, 2017; Drtil, 2023). Modern SAM frameworks seek to optimize software investments by maintaining accurate inventories, monitoring software installations, and ensuring compliance with contractual licensing agreements. These practices provide organizations with improved visibility into software consumption while reducing financial and legal risks associated with software misuse.

Recent developments indicate that Software Asset Management is becoming increasingly automated through intelligent discovery, rule-based classification, and data-driven analysis. Emerging research explores the application of advanced software detection mechanisms and artificial intelligence to automate software identification, entitlement verification, and compliance reporting (Drtil, 2023). Similarly, practitioners increasingly recognize that intelligent asset management systems can improve organizational decision-making by providing continuous insight into software deployment and utilization patterns.

Although Software Asset Management has significantly improved organizational governance, its emphasis remains predominantly administrative. Most SAM solutions focus on identifying what software is installed, whether licenses comply with contractual obligations, and how procurement can be optimized. Comparatively less attention has been devoted to understanding runtime application behavior or dynamically optimizing software utilization during execution. Consequently, current SAM practices provide limited support for reclaiming floating software licenses from application sessions that remain operational but are no longer actively supporting engineering activities.

2.3 SOFTWARE LICENSING AND COMPLIANCE RESEARCH

Software licensing has attracted considerable research attention owing to the increasing complexity of enterprise software ecosystems and the financial consequences of licensing non-compliance. Existing studies have investigated licensing models, contractual obligations, organizational governance, software entitlement management, and automated compliance verification (Alspaugh et al., 2010; Machal-Fulks & Barnett, 2012). These studies collectively emphasize that effective license management requires both technical controls and organizational policies to ensure that software usage remains consistent with vendor agreements.

Recent work has further explored automated approaches for software license compliance, demonstrating that software inventories, behavioral observations, and monitoring tools can improve the detection of unauthorized software usage while reducing manual auditing effort (Al-Luhaidan et al., 2024; Wintersgill et al., 2025). These approaches contribute significantly to improving software governance and reducing compliance risks. However, their primary objective is verifying whether software is used according to licensing terms rather than maximizing the operational efficiency of floating license allocation.

Another important limitation concerns the interpretation of software activity. Existing compliance solutions generally assume that a running application represents legitimate software usage. In practice, however, engineering applications may remain open for extended periods while users perform unrelated tasks. As a result, application execution alone cannot reliably indicate productive utilization of scarce floating licenses. This distinction highlights the need to complement traditional compliance-oriented approaches with mechanisms capable of interpreting runtime activity when making license management decisions.

2.4 RUNTIME MONITORING AND BEHAVIORAL ANALYSIS

Runtime monitoring has become an essential technique for evaluating software performance, diagnosing failures, detecting anomalies, and supporting operational decision-making. Modern monitoring frameworks continuously collect execution metrics such as processor utilization, memory consumption, process state, thread activity, and application events to improve system reliability and operational visibility (Hasselbring & van Hoorn, 2020; Gaol et al., 2022). These techniques are widely adopted across enterprise computing environments because they enable administrators to identify performance bottlenecks and maintain system availability.

Beyond infrastructure monitoring, software engineering research has increasingly recognized that runtime behavior provides valuable information unavailable through static analysis. Studies investigating behavioral evolution demonstrate that execution characteristics frequently reveal functional changes that are difficult to identify from source code alone (Danglot et al., 2020). Likewise, research on software licensing indicates that behavioral information can improve the detection of abnormal software usage patterns beyond conventional inventory-based methods (Wintersgill et al., 2025).

Nevertheless, existing monitoring frameworks primarily evaluate software from the perspective of performance, reliability, or compliance rather than software resource optimization. Most monitoring systems identify whether an application is executing successfully but do not determine whether that application is actively contributing to productive engineering work. Consequently, although runtime monitoring provides the technical foundation for observing application behavior, additional interpretation mechanisms are required before monitoring information can support automated floating-license management.

2.5 RESEARCH GAP

The reviewed literature demonstrates substantial progress across four closely related research domains: enterprise software licensing, Software Asset Management, software license compliance, and runtime software monitoring. Research on enterprise licensing has established effective governance mechanisms for managing software entitlements and contractual obligations (Machal-Fulks & Barnett, 2012; Wikberg, 2010). Software Asset Management has significantly improved organizational visibility into software deployment and lifecycle governance (Phillips, 2017; David, 2023). Runtime monitoring research has developed sophisticated techniques for observing software execution and supporting operational decision-making (Hasselbring & van Hoorn, 2020; Gaol et al., 2022). Meanwhile, software compliance studies have strengthened automated approaches for identifying licensing violations and improving software governance (Al-Luhaidan et al., 2024; Wintersgill et al., 2025).

Despite these advances, an important gap remains. Existing studies generally treat software licensing, runtime monitoring, and software asset management as separate research problems. Administrative solutions focus on software ownership and compliance, whereas monitoring frameworks emphasize infrastructure performance and operational health. Very limited attention has been devoted to integrating runtime activity monitoring with client-side application management to improve floating-license utilization in enterprise engineering environments.

Furthermore, the existing literature provides little guidance on preserving user sessions while reclaiming unused software licenses. Conventional approaches frequently rely on administrative policies, manual intervention, or application termination, each of which may interrupt engineering workflows or discourage users from releasing software resources voluntarily. To the best of the authors' knowledge, there is limited published research investigating runtime activity-aware suspension of inactive CAD applications as a mechanism for transparent floating-license reclamation while maintaining application state.

Accordingly, this study addresses the identified gap by proposing a runtime activity-aware framework that continuously evaluates application execution characteristics to distinguish functionally inactive CAD sessions from actively used applications. Rather than treating software licensing solely as an administrative problem, the proposed framework integrates runtime activity monitoring with application state preservation to enable automatic reclamation of floating licenses while minimizing disruption to engineering workflows. In doing so, the framework establishes a practical connection between Software Asset Management, runtime monitoring, and enterprise engineering operations, contributing a novel approach to improving floating-license utilization in industrial CAD environments.

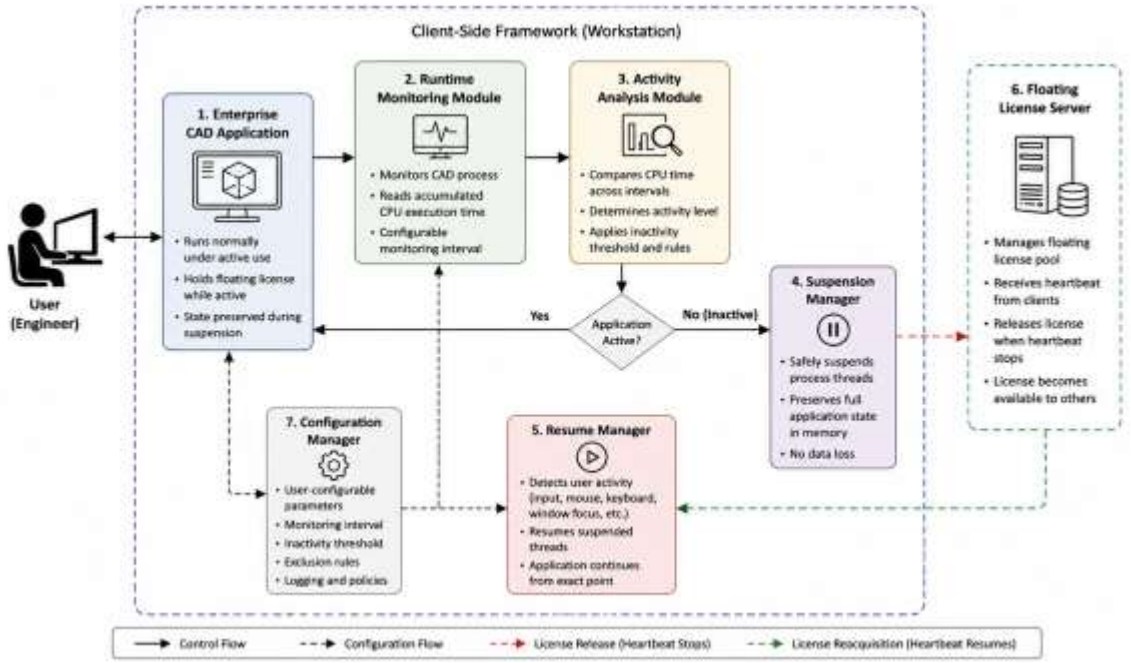
SYSTEM ARCHITECTURE

3.1 OVERALL SYSTEM ARCHITECTURE

The proposed framework is designed to improve the utilization of floating software licenses in enterprise Computer-Aided Design (CAD) environments through continuous runtime activity monitoring and automated license reclamation. Rather than modifying commercial CAD software or

altering the operation of the centralized license server, the framework operates as a lightweight client-side runtime management layer that continuously observes application execution and identifies periods of functional inactivity. When prolonged inactivity is detected, the framework temporarily suspends application execution, allowing the associated floating license to be automatically released by the license server while preserving the complete application state. When the engineer resumes work, the application is restored and transparently reacquires the required license, enabling uninterrupted continuation of the engineering task.

Figure 1: Overall architecture of the proposed runtime activity-aware floating license reclamation framework.



Unlike conventional license management approaches that rely primarily on administrative policies or license server configuration, the proposed framework performs runtime decision-making directly on the engineering workstation. This architectural choice allows inactivity to be evaluated using the operational characteristics of individual CAD applications rather than generalized server-side timeout policies. Consequently, license reclamation decisions become sensitive to actual application execution behavior while remaining independent of modifications to commercial software or proprietary licensing infrastructure.

Figure 1 illustrates the overall architecture of the proposed runtime activity-aware floating license reclamation framework. The architecture consists of six principal components: (1) the enterprise CAD application, (2) the Runtime Monitoring Module, (3) the Activity Analysis Module, (4) the Suspension Manager, (5) the Resume Manager, and (6) the centralized Floating License Server. These components cooperate to monitor application execution, determine runtime activity, reclaim idle licenses, and restore engineering sessions whenever user activity resumes.

Unlike traditional software asset management systems that primarily provide organizational visibility into software usage, the proposed architecture introduces an operational decision layer capable of responding dynamically to runtime conditions. The framework therefore complements existing Software Asset Management and enterprise licensing systems by optimizing the utilization of software licenses during application execution rather than solely managing software assets from an administrative perspective.

The architecture was designed according to four primary objectives:

Transparency, ensuring that engineers can continue their work without modifying established workflows.

State preservation, allowing inactive applications to be suspended without terminating processes or losing engineering data.

Compatibility, enabling deployment without modification of commercial CAD applications or existing floating-license infrastructure.

Operational efficiency, maximizing license availability while minimizing computational overhead and user disruption.

Collectively, these design objectives enable the framework to function as an independent runtime management layer that enhances existing enterprise licensing environments without introducing additional complexity into engineering workflows.

3.2 ARCHITECTURAL COMPONENTS

The proposed framework adopts a modular architecture in which each component performs a dedicated function within the overall license reclamation process. The modular design improves maintainability, facilitates future extensions, and enables each component to operate independently while contributing to the overall runtime activity-aware decision process.

3.2.1 ENTERPRISE CAD APPLICATION

The Enterprise CAD Application represents the engineering software whose floating license utilization is managed by the proposed framework. Typical examples include commercial three-dimensional modeling and plant engineering applications deployed under concurrent licensing schemes. The framework treats the CAD application as an external executable and therefore requires no modification to its source code, executable files, licensing mechanism, or configuration.

When the CAD application is launched, it requests a floating license from the centralized license server using the standard licensing procedure implemented by the software vendor. Once authorization has been granted, the application executes normally while periodically maintaining communication with the license server. Throughout this process, the proposed framework operates independently in the background without interfering with normal engineering activities.

This design choice is particularly important because commercial engineering software is proprietary and typically cannot be modified by end-user organizations. By avoiding changes to application binaries or licensing mechanisms, the proposed framework remains vendor-independent and can potentially be adapted to different floating-license environments with minimal changes.

3.2.2 RUNTIME MONITORING MODULE

The Runtime Monitoring Module constitutes the primary sensing component of the proposed framework. Its responsibility is to continuously observe the execution characteristics of designated CAD processes throughout their operational lifetime.

Unlike conventional monitoring systems that focus on overall processor utilization or infrastructure health, the Runtime Monitoring Module concentrates exclusively on application-level execution characteristics relevant to license utilization. During operation, the module periodically identifies target CAD processes and records their accumulated processor execution time over a configurable monitoring interval. Rather than relying on instantaneous processor usage, the framework evaluates changes in accumulated execution time, providing a more stable representation of application activity across consecutive monitoring periods.

The monitoring interval is configurable, allowing organizations to adapt the framework according to operational requirements and engineering workflows. Shorter intervals enable faster identification of inactive applications, whereas longer intervals reduce monitoring frequency and further minimize computational overhead. Because the monitoring process consists primarily of lightweight operating-system queries, its execution introduces negligible resource consumption relative to the computational demands of enterprise CAD applications.

Another important characteristic of the Runtime Monitoring Module is its continuous background execution. Engineers are not required to initiate monitoring manually or modify their interaction with the CAD application. Once the framework is activated, monitoring proceeds automatically throughout the application lifecycle until the engineering session is terminated.

3.2.3 ACTIVITY ANALYSIS MODULE

The Activity Analysis Module is responsible for transforming raw execution measurements into operational decisions regarding application activity.

Instead of interpreting application activity using simple indicators such as process existence or instantaneous processor utilization, the module evaluates accumulated processor execution characteristics over consecutive monitoring intervals. By comparing execution activity across successive observations, the framework distinguishes applications exhibiting sustained computational activity from those that have remained functionally inactive for a predefined period.

The module incorporates configurable decision thresholds that determine whether an application should continue normal execution or become eligible for license reclamation. These thresholds can be adjusted according to organizational policies, allowing engineering organizations to balance rapid license recovery against the risk of interrupting legitimate engineering activities.

Importantly, the Activity Analysis Module does not immediately reclaim licenses whenever reduced execution activity is detected. Instead, it evaluates runtime behavior continuously to ensure that temporary fluctuations in application execution do not result in unnecessary suspension decisions. This conservative decision strategy improves operational reliability while reducing the likelihood of reclaiming licenses during short periods of legitimate inactivity.

By separating activity interpretation from execution monitoring, the proposed architecture improves modularity and allows future enhancements to incorporate additional behavioral indicators, machine-learning techniques, or adaptive decision policies without requiring modifications to the remaining framework components.

3.2.4 SUSPENSION MANAGER

The Suspension Manager is responsible for temporarily suspending CAD applications that have been identified by the Activity Analysis Module as functionally inactive. Rather than terminating the application or forcing the user to close the software, this component preserves the complete execution state of the running process by suspending all executable threads associated with the application. Consequently, the application remains resident in system memory while ceasing active execution.

Suspending application execution serves two important purposes. First, it prevents inactive applications from unnecessarily consuming computational resources while preserving all user-generated data, graphical models, and application context. Second, because the suspended application no longer maintains normal communication with the floating license server, the associated heartbeat eventually expires according to the license server configuration. As a result, the floating license is automatically returned to the available license pool without requiring explicit interaction with the license management infrastructure.

An important design characteristic of the Suspension Manager is that it operates entirely on the client workstation. The framework neither modifies the commercial CAD application nor communicates directly with the license server to release licenses. Instead, license reclamation occurs naturally through the existing licensing mechanism after application execution has been suspended. This design preserves compatibility with commercial licensing systems while avoiding dependencies on vendor-specific application programming interfaces (APIs) or proprietary licensing protocols.

The Suspension Manager also incorporates safeguards that prevent repeated suspension requests from being issued to applications that have already entered the suspended state. This ensures stable framework operation while avoiding unnecessary processing during subsequent monitoring intervals.

3.2.5 APPLICATION RECOVERY MODULE

The Application Recovery Module restores suspended engineering sessions whenever the user wishes to continue working with the CAD application. Unlike traditional approaches that require applications to be restarted after license release, the proposed framework preserves the complete execution context throughout the suspension period, allowing recovery to occur from the exact point at which execution was previously suspended.

Upon user request, the Application Recovery Module resumes all suspended application threads, allowing the CAD application to continue normal execution. As application execution resumes, communication with the floating license server is automatically re-established through the application's standard licensing mechanism. If a floating license is available within the centralized license pool, the application transparently reacquires the required license and continues operation without requiring

the user to reopen project files or reconstruct the engineering workspace.

This approach significantly reduces workflow interruption compared with conventional license management strategies based on application termination or manual software restart. Engineers can therefore temporarily release software licenses during periods of inactivity while maintaining continuity of their engineering tasks.

Separating recovery functionality into an independent architectural component also improves modularity. Future versions of the framework could support alternative recovery mechanisms, automated restoration policies, or integration with enterprise workflow management systems without requiring modifications to the monitoring or suspension components.

3.2.6 FLOATING LICENSE SERVER INTERACTION

The Floating License Server represents the external licensing infrastructure responsible for allocating and managing concurrent software licenses within the enterprise environment. The proposed framework treats the license server as an independent system and does not require administrative access or modification of its internal operation.

During normal application execution, the CAD software periodically exchanges heartbeat messages with the license server to maintain ownership of the allocated floating license. As long as these communications continue successfully, the license remains reserved for the corresponding application session.

Following application suspension, heartbeat communication gradually ceases because application execution has stopped. After the predefined timeout configured by the licensing system expires, the server interprets the missing heartbeat as termination of the application session and automatically returns the floating license to the shared license pool. From the perspective of the license server, this behavior follows the standard license management process without requiring additional software components or customized server-side policies.

Similarly, when the Application Recovery Module restores application execution, the CAD application resumes normal communication with the license server. The application subsequently requests and reacquires a floating license using the vendor's existing licensing protocol. Because the proposed framework relies entirely on standard licensing behavior, it remains compatible with existing enterprise deployment environments while minimizing implementation complexity.

3.2.7 CONFIGURATION MANAGER

To support deployment across diverse engineering environments, the proposed framework includes a Configuration Manager responsible for controlling operational parameters without requiring modification of the application source code.

The Configuration Manager maintains user-defined settings including the monitoring interval, inactivity threshold, monitored CAD process names, logging preferences, and operational policies. These parameters enable organizations to adapt the framework according to software characteristics, engineering workflows, and organizational requirements.

Externalizing configuration information provides two important advantages. First, system administrators can optimize framework behavior for different engineering applications without recompiling the software. Second, configurable operational policies simplify maintenance and facilitate deployment across multiple engineering workstations using consistent configuration profiles.

The modular separation of configuration management from runtime monitoring also improves maintainability by isolating operational parameters from application logic, thereby simplifying future enhancements and reducing implementation complexity.

3.3 OPERATIONAL WORKFLOW

Figure 2 illustrates the overall operational workflow of the proposed framework. The workflow begins when an engineer launches a commercial CAD application that requests a floating license from the centralized license server. After successful license allocation, the application executes normally while the Runtime Monitoring Module continuously observes its execution characteristics throughout the engineering session.

At predefined monitoring intervals, the Runtime Monitoring Module records the accumulated processor execution time of the monitored CAD process and forwards the collected information to the Activity Analysis Module. The Activity Analysis Module evaluates changes in execution activity over successive monitoring periods and determines whether the application remains actively engaged in engineering work.

If sufficient runtime activity is detected, the application continues normal execution and monitoring proceeds during the next observation interval. No additional intervention is performed by the framework, ensuring complete transparency during active engineering work.

Conversely, if application execution remains below the configured activity threshold for the specified monitoring period, the Activity Analysis Module classifies the application as functionally inactive and issues a suspension request to the Suspension Manager. The Suspension Manager subsequently suspends all executable threads associated with the CAD process while preserving the complete application state in memory. Because application execution has ceased, heartbeat communication with the floating license server eventually stops. After expiration of the license timeout configured by the licensing system, the server automatically returns the floating license to the enterprise license pool, making it immediately available for allocation to other engineers.

When the original user wishes to continue working, the Application Recovery Module restores execution by resuming the suspended application threads. The CAD application subsequently re-establishes communication with the floating license server and transparently reacquires a floating license using the standard licensing procedure. From the user's perspective, engineering work continues from the previously suspended state without reopening files or repeating completed operations.

This operational workflow enables automatic redistribution of floating software licenses while preserving engineering continuity and minimizing disruption to normal design activities.

3.4 ARCHITECTURAL CHARACTERISTICS

Several architectural principles guided the design of the proposed framework to ensure compatibility with enterprise engineering environments.

Client-side deployment enables implementation without modifying commercial CAD software or centralized license server infrastructure. This significantly reduces deployment complexity and facilitates integration with existing engineering systems.

Transparency ensures that monitoring and license reclamation occur without altering normal engineering workflows. Engineers continue using familiar CAD applications while the framework operates unobtrusively in the background.

Application state preservation distinguishes the proposed framework from conventional approaches based on application termination. By suspending rather than closing applications, the framework maintains all user data, execution context, and engineering progress throughout the license reclamation process.

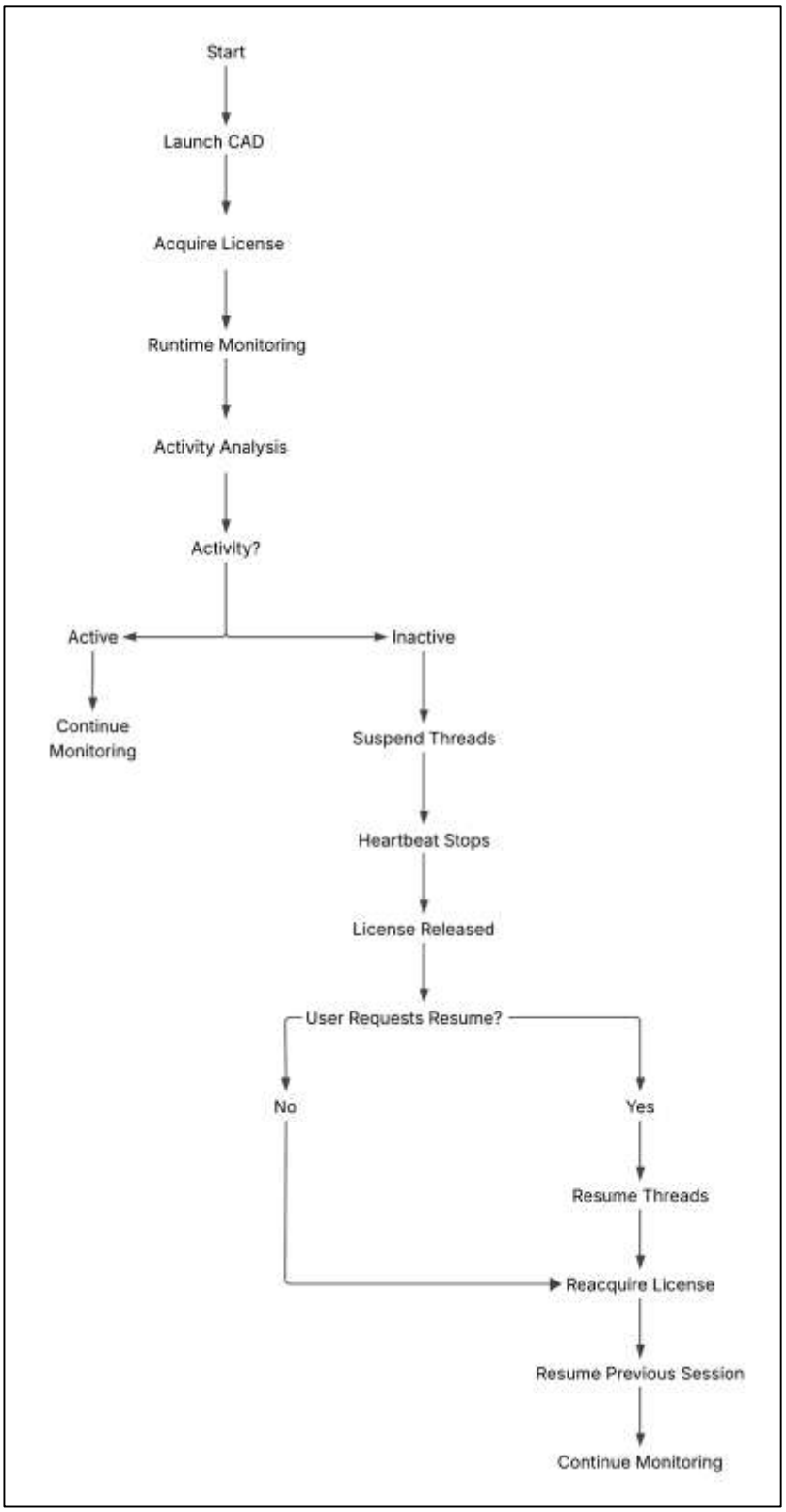
Vendor independence improves framework portability because license management relies exclusively on the standard behavior of existing floating-license systems. Consequently, the framework does not require proprietary licensing APIs or software modifications, making adaptation to different commercial CAD environments more straightforward.

Finally, the modular organization of the framework enhances maintainability, extensibility, and reliability. Individual architectural components can be modified or extended independently without affecting the remaining framework, thereby supporting future enhancements such as adaptive activity thresholds, machine learning-based activity classification, or integration with enterprise software asset management platforms.

The architectural design presented in this section establishes the structural foundation for the proposed runtime activity-aware framework. Building upon this architecture, the following section presents the

methodology employed to monitor runtime activity, identify functionally inactive application sessions, and perform dynamic floating-license reclamation through controlled suspension and application recovery.

Figure 2: Operational workflow of the proposed runtime activity-aware floating license reclamation framework.



PROPOSED METHODOLOGY

4.1 METHODOLOGY OVERVIEW

The proposed runtime activity-aware framework employs a continuous monitoring strategy to identify functionally inactive CAD application sessions and dynamically reclaim floating software licenses without disrupting engineering workflows. Unlike conventional license management approaches that rely on manual user intervention or application termination, the proposed methodology continuously evaluates the runtime execution characteristics of CAD applications to determine whether an allocated floating license is actively contributing to productive engineering work.

Figure 3: Runtime activity decision model illustrating periodic monitoring, activity evaluation, threshold comparison, and subsequent execution path.

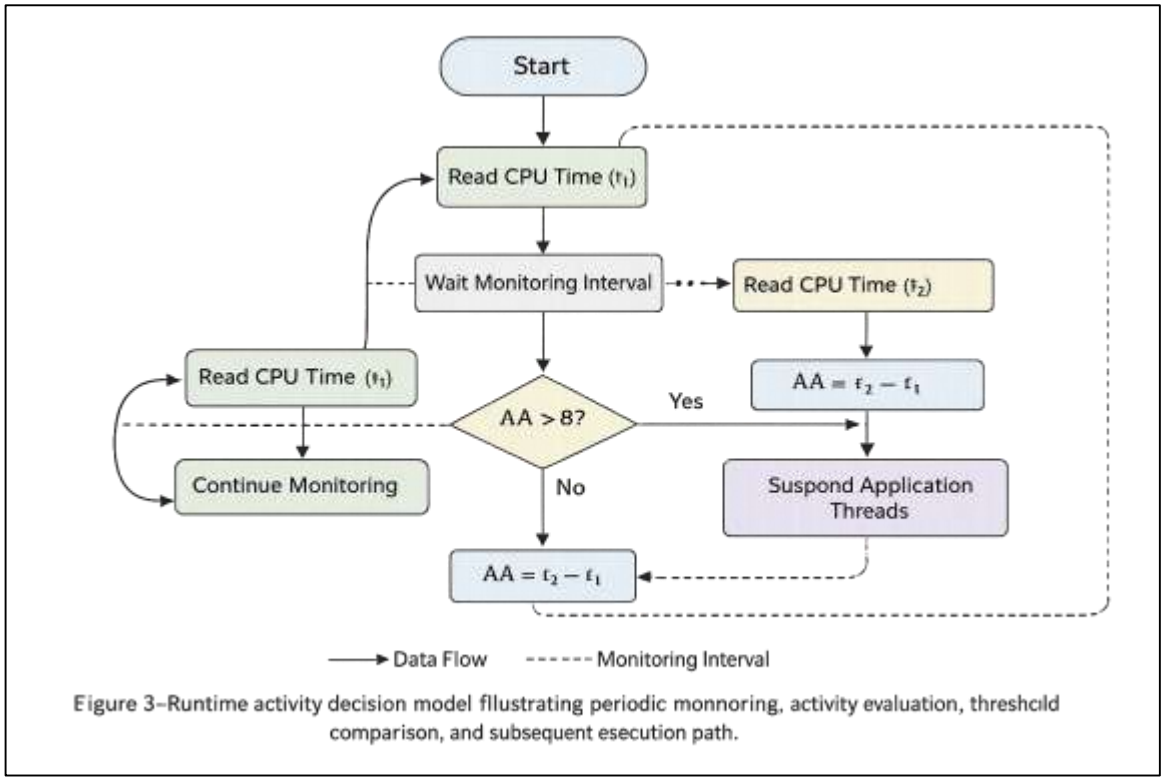


Figure 3–Runtime activity decision model fllustrating periodic monnoring, activity evaluation, threshcild comparison, and subsequent esecution path.

The methodology is based on the observation that engineering applications exhibit measurable processor execution activity during periods of active use. Operations such as three-dimensional model creation, feature editing, simulation, rendering, constraint solving, and file manipulation require continuous processor execution that accumulates over time. Conversely, applications left idle for extended periods generate little or no additional processor execution despite continuing to reserve valuable floating licenses.

Rather than monitoring user input devices or relying on operating system idle timers, the proposed framework evaluates application activity directly from the execution behavior of the monitored process. This design provides a more application-centric representation of runtime activity because processor execution reflects the computational workload generated by the engineering software itself instead of indirect indicators of user presence. Consequently, temporary periods during which users observe simulation results or review complex models without interacting with input devices are less likely to be incorrectly classified as inactive.

The methodology consists of four sequential phases that execute continuously throughout the lifetime of each monitored CAD application. First, the Runtime Monitoring Module periodically records the accumulated processor execution time of the target process. Second, the Activity Analysis Module compares successive execution measurements to quantify application activity during the monitoring

interval. Third, if the measured execution activity falls below a predefined threshold, the Suspension Manager temporarily suspends the application while preserving its complete execution state. Finally, when the user elects to continue working, the Application Recovery Module restores application execution, allowing the software to transparently reacquire a floating license using the standard licensing mechanism. Because each phase operates independently while exchanging only well-defined execution information, the methodology remains modular, extensible, and compatible with a wide variety of commercial CAD applications and floating licensing infrastructures.

4.2 RUNTIME ACTIVITY MONITORING

The Runtime Monitoring Module forms the foundation of the proposed methodology by continuously observing the execution characteristics of monitored CAD applications throughout their operational lifetime. The primary objective of this component is to collect quantitative runtime information that accurately reflects the computational activity of each application while introducing minimal overhead to the engineering workstation.

The monitoring process begins immediately after a supported CAD application is detected within the operating system. The framework identifies the target process using configurable process definitions stored within the Configuration Manager, enabling multiple engineering applications to be monitored without modifying the monitoring logic. Once the target process has been identified, the monitoring module establishes a periodic observation cycle governed by a configurable monitoring interval.

At the beginning of each observation cycle, the framework records the cumulative processor execution time associated with the monitored process. In the Windows operating system, this information is obtained through the Process.UserProcessorTime property, which represents the total amount of processor time consumed by the application's user-mode threads since process creation.

Unlike instantaneous processor utilization measurements, accumulated processor execution time provides a monotonically increasing measure of computational work performed by the application. Consequently, transient fluctuations in processor utilization do not significantly influence activity assessment, resulting in a more stable representation of application behavior over extended monitoring intervals.

Let

$$U(t)$$

represent the accumulated processor execution time of the monitored process at observation time t. The accumulated execution activity is therefore expressed as

$$\boxed{A(t) = U(t)} \qquad (1)$$

where

denotes the accumulated execution activity,

represents the accumulated processor execution time reported by the operating system.

Equation (1) forms the basis for all subsequent activity analysis performed by the framework.

During each monitoring cycle, two execution measurements are obtained. The first measurement is captured at the beginning of the monitoring interval, after which the framework waits for the configured observation period before collecting a second measurement. Because accumulated processor execution time only increases while the application performs computational work, the difference between successive observations represents the processor activity generated during that interval.

This monitoring strategy offers several practical advantages. First, it avoids continuous processor polling that could unnecessarily increase monitoring overhead. Second, because only two lightweight system calls are required during each monitoring interval, the computational cost of runtime monitoring remains negligible compared with the resource requirements of modern CAD applications. Finally, accumulated execution measurements remain independent of temporary processor scheduling variations, improving the stability of inactivity detection across heterogeneous workstation

configurations.

The monitoring interval itself represents an important configurable parameter within the proposed framework. Short monitoring intervals allow inactive applications to be identified more rapidly but may increase the frequency of runtime observations. Conversely, longer monitoring intervals reduce monitoring activity while delaying license reclamation. Externalizing this parameter through the Configuration Manager enables organizations to balance responsiveness and monitoring overhead according to their operational requirements.

4.3 ACTIVITY DECISION MODEL

Following acquisition of successive execution measurements, the Activity Analysis Module determines whether the monitored application remains functionally active. Rather than evaluating absolute processor execution time, the framework measures the increase in accumulated execution activity observed during the monitoring interval.

Let

$$U(t_1)$$

represent the accumulated processor execution time recorded at the beginning of the monitoring interval, and

$$U(t_2)$$

represent the accumulated execution time measured after the monitoring interval has elapsed.

The processor execution activity generated during the monitoring interval is computed as

$$\boxed{\Delta U = U(t_2) - U(t_1)} \qquad (2)$$

where

- ΔU denotes processor execution activity during the observation interval.

The calculated activity value is subsequently compared with a predefined execution threshold θ. This threshold represents the minimum processor execution required for an application to be considered functionally active.

The activity classification rule is defined as

$$\boxed{\text{State} = \begin{cases} \text{Active}, & \Delta U > \theta \\ \text{Inactive}, & \Delta U \leq \theta \end{cases}} \qquad (3)$$

If the measured activity exceeds the threshold, the application is considered to be actively performing computational work and monitoring continues during the next observation interval. Conversely, when processor execution remains below the configured threshold, the application is classified as functionally inactive, and the Activity Analysis Module initiates the dynamic license reclamation procedure described in the following subsection.

This threshold-based decision model was selected because of its simplicity, computational efficiency, and ease of configuration. Unlike statistical or machine learning approaches that require training data and parameter optimization, the proposed decision model can be deployed immediately within enterprise environments while remaining fully interpretable by system administrators. Furthermore, the modular architecture allows future research to replace the threshold comparison mechanism with more sophisticated activity classifiers without modifying the surrounding monitoring and recovery framework.

Algorithm 1. Runtime activity monitoring and activity classification procedure.

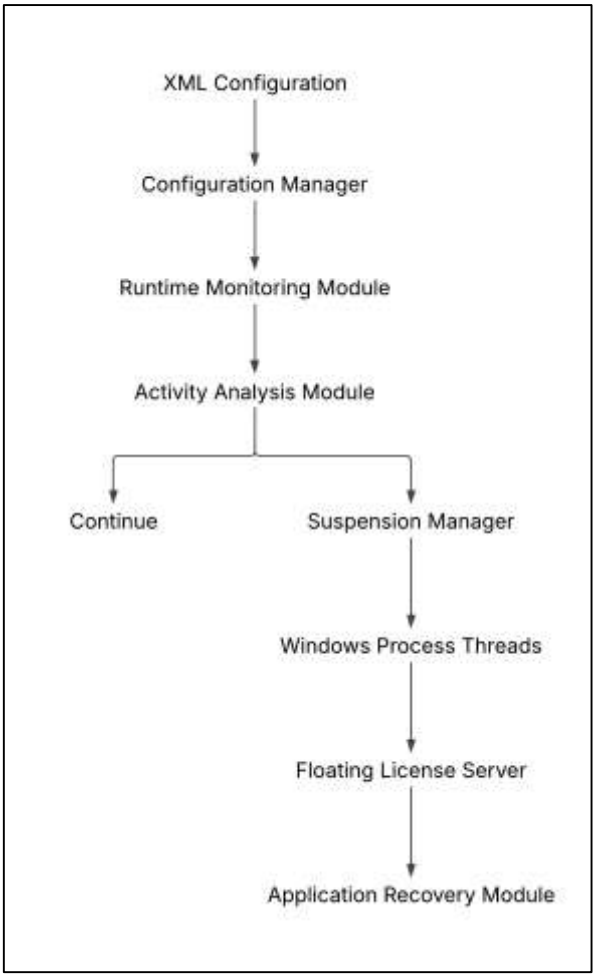
Input: Monitored CAD process P, Monitoring interval T, Activity threshold θ
1: while P is running do 2: U1 ← UserProcessorTime(P) 3: Wait(T) U2 ← UserProcessorTime(P) 4: ΔU ← U2 − U1 5: if ΔU > θ then 6: Continue monitoring 7: else 8: Invoke Suspension Manager 9: end if 10: end while

IMPLEMENTATION

The proposed runtime activity-aware framework was implemented as a client-side application using the Microsoft .NET platform and the C# programming language. The implementation targets Microsoft Windows operating systems, which represent the predominant deployment environment for commercial CAD applications used in enterprise engineering organizations. The framework operates independently of the monitored CAD software and requires no modification of either the engineering applications or the centralized floating license server.

The software follows a modular implementation consistent with the architectural design presented in Section 3. Individual functional responsibilities are encapsulated within dedicated software components responsible for process monitoring, activity analysis, application suspension, session recovery, configuration management, and operational logging. This modular organization improves maintainability while allowing individual components to be updated or extended without affecting the remainder of the framework.

Figure 4: Software implementation architecture of the proposed framework.



When the framework is launched, configuration parameters are first loaded from an external XML configuration file. These parameters define the CAD processes to be monitored, the runtime monitoring interval, activity threshold values, logging preferences, and additional operational settings. Externalizing these parameters eliminates the need to recompile the application when deployment requirements change and allows administrators to customize framework behavior for different engineering environments.

Following initialization, the framework continuously monitors the Windows process list for supported CAD applications. Process identification is performed using the executable names specified within the configuration file. Once a target application is detected, the Runtime Monitoring Module establishes an independent monitoring cycle for the corresponding process.

Runtime activity is evaluated using the Process.UserProcessorTime property provided by the .NET runtime. This operating system metric reports the cumulative processor execution time consumed by the user-mode threads of a running process since its creation. At the beginning of each monitoring interval, the framework records the accumulated processor execution time of the monitored application. After waiting for the configured monitoring period, a second measurement is obtained, and the difference between successive values is computed to quantify processor activity during that interval. This implementation directly realizes the monitoring methodology described in Section 4 while avoiding continuous processor utilization sampling.

To minimize runtime overhead, the monitoring component performs lightweight periodic observations rather than continuous polling. Each monitoring cycle requires only two operating system queries separated by the configured monitoring interval, allowing the framework to execute with negligible computational impact relative to the resource demands of modern CAD applications. Because the framework observes only selected engineering applications, unnecessary processing of unrelated system processes is avoided.

When the measured processor execution activity falls below the configured threshold, the Activity Analysis Module classifies the application as functionally inactive and requests intervention from the Suspension Manager. The Suspension Manager enumerates all execution threads associated with the target process and temporarily suspends each thread while preserving the application's complete execution state in memory. Since the application is neither terminated nor unloaded, all open documents, graphical models, user settings, and internal execution context remain intact throughout the suspension period.

A key implementation characteristic is that the framework performs no direct interaction with the floating license server. Instead, license reclamation relies entirely on the standard behavior of commercial floating-license systems. Once application execution has been suspended, the software no longer transmits periodic heartbeat messages to the license server. After expiration of the server's predefined timeout interval, the floating license is automatically returned to the shared license pool using the licensing system's native timeout mechanism. Consequently, the framework remains independent of proprietary licensing APIs while preserving compatibility with existing enterprise licensing infrastructures.

Application restoration is performed through the Application Recovery Module. When the user elects to resume engineering work, the recovery component restores execution by resuming all previously suspended application threads. The CAD application subsequently re-establishes communication with the floating license server using its standard licensing protocol. If a floating license is available, the application transparently reacquires the license and continues execution from the previously preserved application state. Because neither application restart nor project reloading is required, engineers can continue working without reconstructing their design environment or repeating previously completed operations.

To improve operational reliability, the implementation incorporates several safeguards that prevent inconsistent application states. Before initiating suspension, the framework verifies that the target process is still executing and has not already been suspended by a previous monitoring cycle. Similarly, recovery operations are applied only to applications that have been explicitly identified as suspended by the framework. These validation procedures reduce the likelihood of redundant operations and

contribute to stable long-term execution in enterprise environments where multiple engineering applications may be monitored simultaneously.

The implementation also includes a configurable logging facility that records significant runtime events throughout framework operation. Typical log entries include application detection, monitoring initialization, measured activity values, suspension events, license reclamation intervals, recovery operations, and execution errors. These records facilitate debugging, performance analysis, and operational auditing while providing administrators with visibility into framework behavior during enterprise deployment.

Overall, the implementation faithfully realizes the proposed runtime activity-aware methodology using standard Windows operating system services and the .NET runtime environment. By combining lightweight runtime monitoring, modular software organization, controlled thread suspension, and transparent application recovery, the framework enables dynamic floating-license reclamation without requiring modifications to commercial CAD software or centralized licensing infrastructure. This implementation provides the practical foundation for the experimental evaluation presented in the following section.

EXPERIMENTAL EVALUATION AND PERFORMANCE ANALYSIS

6.1 EXPERIMENTAL ENVIRONMENT

The proposed runtime activity-aware framework was evaluated in a production enterprise engineering environment supporting collaborative three-dimensional plant design using commercial CAD software. The evaluation was conducted within a Citrix-based virtual desktop infrastructure (VDI), where engineering users accessed centralized CAD applications through Windows virtual machines. This deployment model reflects the organization's operational environment and represents a typical enterprise configuration for managing computationally intensive engineering applications and shared floating software licenses.

Each virtual workstation was provisioned with four virtual CPU cores based on an Intel Xeon processor platform operating at approximately 3.3 GHz and 16 GB of memory. The client environment executed Microsoft Windows 10 Enterprise (64-bit) together with Microsoft .NET Framework 4.7, which hosted the client-side implementation of the proposed framework. The monitored engineering applications consisted of AVEVA E3D versions 2.1 and 3.1, specifically the Design and Draw modules, both of which employ concurrent floating licensing managed by AVEVA License Manager 2.1 hosted on Microsoft Windows Server 2022.

The enterprise environment supported approximately 500 engineering users distributed across multiple geographical regions and time zones. Daily engineering activities were performed across approximately 300 Citrix-hosted virtual workstations sharing a centralized floating-license pool of approximately 265 concurrent CAD licenses. Because engineering activities were performed asynchronously across different working hours, concurrent license demand varied throughout the operational day.

The framework operated entirely on the client workstation and required no modification to either the CAD applications or the enterprise licensing infrastructure. Runtime activity was evaluated every 900 s (15 min) using accumulated processor execution time obtained through the Windows operating system. Applications exhibiting less than 500 ms of accumulated processor execution during a monitoring interval were classified as functionally inactive and became candidates for controlled suspension.

The framework underwent operational validation during a four-week production deployment following implementation. To assess its broader operational impact, historical engineering usage reports covering the complete 2018 calendar year together with enterprise license utilization records spanning March 2018 to March 2019 were analyzed. This combination enabled both short-term validation of framework operation and long-term assessment of enterprise license utilization.

Table 1: Experimental hardware, software and framework configuration.

Category	Parameter	Configuration
Deployment Environment	Infrastructure	Enterprise Citrix Virtual desktop infrastructure (VDI)
	Number of virtual workstations	Approximately 300
	Engineering users	Approximately 500 users distributed across multiple geographical regions and time zones
	Framework validation period	4 weeks following production deployment
	Operational data analyzed	January–December 2018 (12 months)
Virtual Machine Configuration	Processor	4 virtual CPUs based on Intel Xeon processor platform (~3.3 GHz)
	Memory	16 GB RAM
	Operating system	Microsoft Windows 10 Enterprise (64-bit)
	Runtime platform	Microsoft .NET Framework 4.7
CAD Environment	CAD software	AVEVA E3D versions 2.1 and 3.1
	Evaluated modules	Design and Draw
	License type	Floating (concurrent) licensing
	Shared license pool	Approximately 265 concurrent CAD licenses
License Management Infrastructure	License management software	AVEVA License Manager 2.1
	License server operating system	Microsoft Windows Server 2022
Framework Configuration	Monitoring interval	900 s (15 min)
	Activity threshold	500 ms accumulated processor execution time
	Monitoring technique	Runtime monitoring using Process.UserProcessorTime
	License reclamation mechanism	Automatic license release through heartbeat timeout following controlled application thread suspension
	Recovery mechanism	Application Recovery Module resumes suspended threads and enables transparent license reacquisition

6.2 EVALUATION METHODOLOGY

The objective of the experimental evaluation was to determine whether the proposed framework could improve floating-license utilization without disrupting normal engineering workflows. Rather than relying exclusively on framework-generated logs, the evaluation combined two independent enterprise datasets to provide complementary perspectives on framework performance.

The first dataset consisted of engineering software usage reports exported from the enterprise license management infrastructure. These reports contained detailed monthly software usage information, including licensed application features, user sessions, and accumulated engineering usage hours. The dataset comprised more than forty thousand recorded usage entries covering the entire 2018 operational year. Monthly usage hours derived from these records were used to characterize engineering workload before and after deployment.

The second dataset consisted of concurrent license utilization records collected by the organization's License Management Tool. Unlike the proposed framework, which monitors application execution on individual workstations, this enterprise monitoring system continuously records concurrent license allocation across the shared floating-license pool. Consequently, these records provide an independent measure of floating-license consumption and are unaffected by the internal operation of the proposed framework.

The framework was deployed toward the end of July 2018 following the organization's annual summer vacation period. Engineering activity naturally decreases during May, June, and July because of scheduled employee leave, resulting in lower software utilization. This seasonal effect was considered during analysis to avoid attributing changes in license consumption solely to the proposed framework. Therefore, engineering workload and concurrent license utilization were evaluated jointly to distinguish operational improvements from seasonal fluctuations in engineering activity.

The evaluation therefore addressed two complementary research questions:

Did engineering workload recover after the summer period?

Did concurrent floating-license utilization remain lower following deployment despite the recovery in engineering activity?

Positive answers to both questions would indicate that improvements in license utilization were associated with runtime activity-aware license reclamation rather than reduced engineering demand.

6.3 EXPERIMENTAL RESULTS

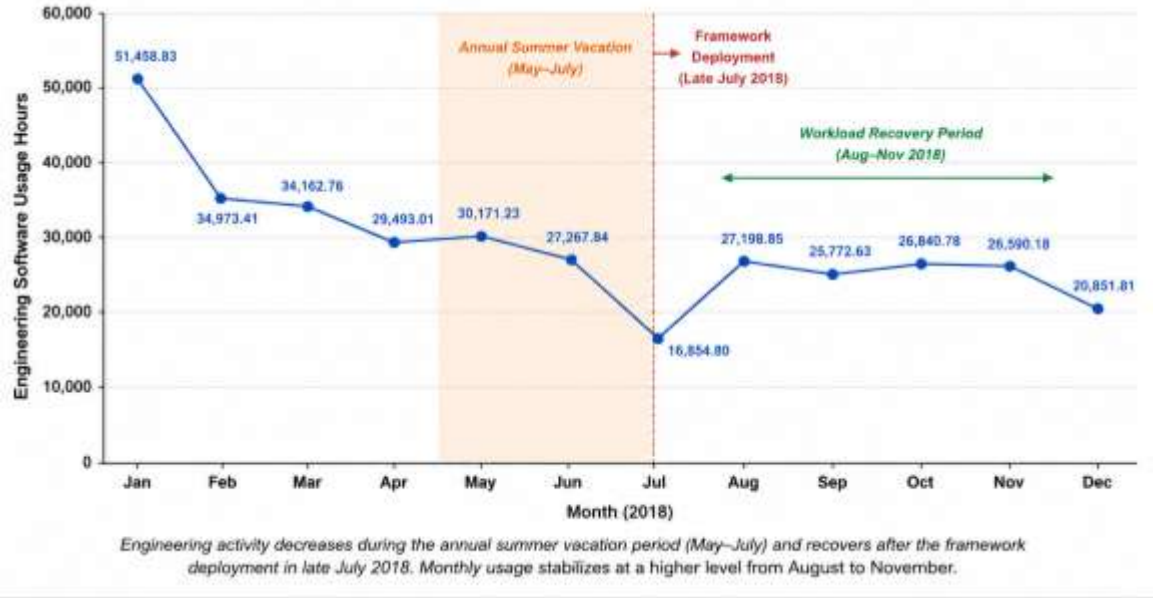
6.3.1 ENGINEERING WORKLOAD ANALYSIS

Figure 5 presents the monthly engineering software usage hours derived from the enterprise licensing reports for the 2018 calendar year. During the first quarter of the year, engineering activity remained consistently high, with January exhibiting the largest accumulated software usage. Activity gradually declined during the second quarter and reached its lowest level during July, corresponding to the organization's annual summer vacation period.

Following deployment of the proposed framework at the end of July, engineering activity increased again during August and remained relatively stable throughout September, October, and November. Monthly engineering usage during these months consistently exceeded twenty-five thousand usage hours, indicating that routine engineering operations had resumed after the seasonal reduction.

These observations demonstrate that the enterprise continued to experience substantial engineering workloads following framework deployment. Consequently, any subsequent reduction in floating-license utilization cannot be explained solely by reduced engineering activity.

Figure 5: Monthly engineering software usage hours during the 2018 evaluation period.



Engineering activity decreases during the annual summer vacation period (May–July) and recovers after the framework deployment in late July 2018. Monthly usage stabilizes at a higher level from August to November.

6.3.2 FLOATING-LICENSE UTILIZATION

Enterprise concurrent license utilization before and after deployment of the proposed framework is illustrated in Figure 6. The figure was generated from independent records collected by the organization's License Management Tool and therefore provides an unbiased representation of enterprise floating-license consumption.

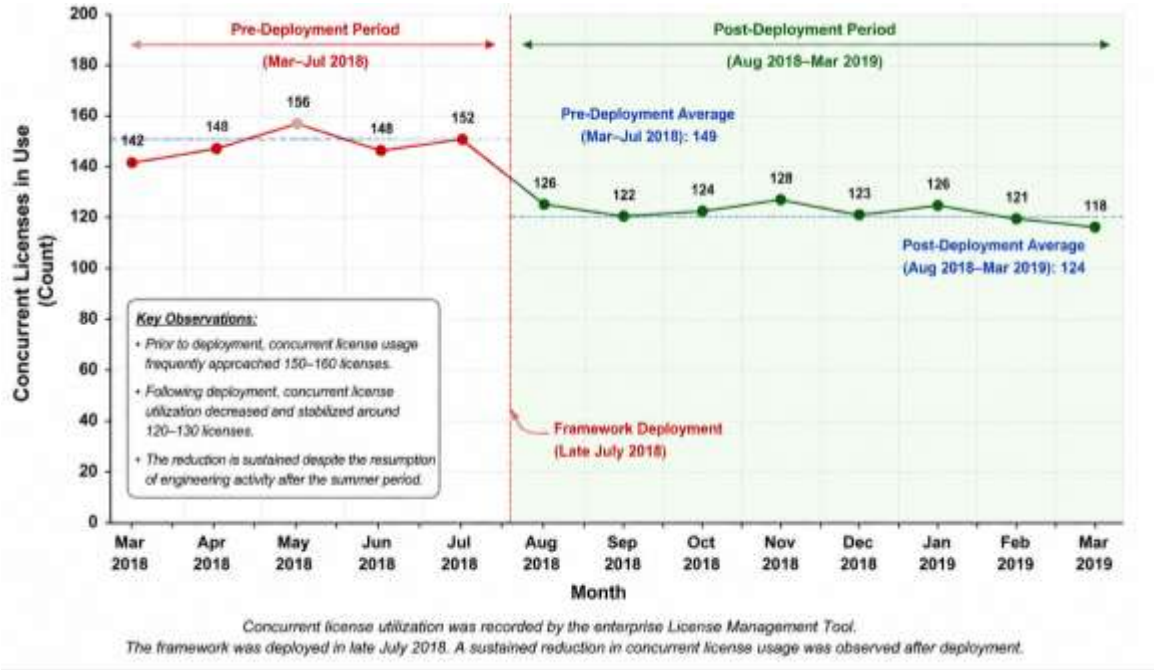
Prior to deployment, Concurrent license utilization was consistently higher before deployment and

remained visibly lower following deployment.. Following deployment in late July 2018, a sustained reduction in concurrent license utilization became evident. During the subsequent operational months, concurrent license consumption generally remained within the range of approximately 120–130 licenses despite engineering activity returning to normal production levels after the summer vacation period.

The reduction in concurrent license utilization is consistent with the intended operation of the proposed framework. Applications exhibiting prolonged periods of negligible computational activity were automatically suspended, allowing their associated floating licenses to return to the enterprise license pool following expiration of the licensing heartbeat timeout. Because suspended applications retained their execution state, engineers were subsequently able to resume their work without restarting the CAD application or reopening project data.

Although multiple operational factors may influence enterprise software utilization, the sustained reduction observed following deployment suggests that runtime activity-aware license reclamation improved the availability of floating licenses for active engineering users.

Figure 6: Enterprise concurrent floating-license utilization before and after deployment of the proposed framework.



Concurrent license utilization was recorded by the enterprise License Management Tool.
The framework was deployed in late July 2018. A sustained reduction in concurrent license usage was observed after deployment.

6.3.3 OPERATIONAL EFFECTIVENESS

A key objective of this study was to distinguish improvements in license utilization from natural fluctuations in engineering workload. Considering either dataset independently would provide only partial evidence. However, evaluating both datasets together enables a more comprehensive interpretation of framework performance.

Engineering usage reports indicate that software usage increased following the summer vacation period and remained stable throughout the final months of the year. At the same time, enterprise license utilization remained consistently lower than during the pre-deployment period. This divergence suggests that engineering productivity was maintained while fewer concurrent floating licenses were required to support daily operations.

From an operational perspective, these observations indicate that licenses previously occupied by inactive application sessions became available for redistribution to other engineers. Consequently, existing license resources were utilized more efficiently without requiring additional software purchases or modifications to the centralized licensing infrastructure.

The independent nature of the two datasets further strengthens these conclusions. Engineering usage reports quantify productive software use, whereas license utilization records quantify resource allocation. The agreement between these independent observations provides strong evidence that the proposed framework achieved its intended objective of improving floating-license utilization while

preserving engineering continuity.

Table 2: Monthly engineering usage hours and corresponding operational phase (Pre-deployment, Summer, Post-deployment).

Month	Engineering Usage Hours	Operational Phase	Remarks
January	51,458.83	Pre-deployment	Highest engineering workload
February	34,973.41	Pre-deployment	Normal engineering operations
March	34,162.76	Pre-deployment	Normal engineering operations
April	29,493.01	Pre-deployment	Normal engineering operations
May	30,171.23	Summer vacation	Seasonal reduction in workforce availability begins
June	27,267.84	Summer vacation	Continued seasonal decline in engineering activity
July	16,854.80	Summer vacation / Deployment	Lowest workload; framework deployed in late July 2018
August	27,198.85	Post-deployment	Engineering activity resumes following deployment
September	25,772.63	Post-deployment	Stable engineering workload
October	26,840.78	Post-deployment	Stable engineering workload
November	26,590.18	Post-deployment	Stable engineering workload
December	20,851.81	Post-deployment	Reduced activity associated with year-end holiday period

Table 2: Monthly engineering software usage hours derived from enterprise licensing records. The table illustrates the transition from the pre-deployment period, through the annual summer vacation period during which the framework was deployed, to the post-deployment operational phase used for evaluating long-term floating-license utilization.

6.4 OPERATIONAL PERFORMANCE ANALYSIS

Beyond improving license utilization, the proposed framework demonstrated stable operation throughout the production deployment. Runtime monitoring required only periodic acquisition of accumulated processor execution time separated by a configurable monitoring interval of fifteen minutes. Consequently, monitoring operations introduced minimal computational activity compared with the resource requirements of modern CAD applications.

The client-side architecture also simplified enterprise deployment. Because the framework operates independently of both the CAD software and the license server, no modifications were required to existing engineering applications, licensing policies, or server infrastructure. This significantly reduced deployment complexity while maintaining compatibility with commercially supported software.

The production deployment further demonstrated the scalability of the proposed approach. The framework operated within an enterprise environment comprising approximately 300 virtual workstations supporting approximately 500 distributed engineering users and a centralized pool of approximately 265 concurrent floating licenses. Throughout the evaluation, suspended applications preserved their complete execution state, enabling engineers to resume work without reopening projects or reconstructing application sessions.

These operational characteristics indicate that runtime activity-aware license reclamation can be integrated into existing enterprise CAD environments with minimal disruption while providing measurable improvements in floating-license utilization.

6.5 DISCUSSION OF RESULTS

The experimental evaluation demonstrates that the proposed framework can improve floating-license utilization under real production conditions without altering existing engineering workflows. Analysis of engineering usage reports confirms that production activity recovered following the annual summer vacation period, while independent license utilization records show that concurrent floating-license consumption remained lower after framework deployment. Together, these observations support the conclusion that the observed reduction in license consumption is consistent with more efficient utilization of existing floating-license resources rather than a reduction in engineering activity.

A notable strength of the evaluation is the use of independent enterprise datasets collected over an extended operational period. Combining engineering workload records with license utilization data provides a more comprehensive assessment than relying solely on framework-generated logs. Nevertheless, the study is limited to a single enterprise deployment and a specific commercial CAD

platform. Future work will investigate adaptive activity thresholds, multi-application support, and evaluation across additional engineering organizations and licensing environments.

DISCUSSION

7.1 INTERPRETATION OF FINDINGS

The experimental evaluation demonstrates that the proposed runtime activity-aware framework can improve floating-license utilization within a production enterprise CAD environment while preserving engineering workflow continuity. Unlike conventional license management approaches that rely on user intervention or predefined inactivity policies, the proposed framework evaluates application activity based on accumulated processor execution time. This enables license reclamation decisions to be driven by the actual computational activity of the application rather than indirect indicators such as keyboard or mouse inactivity.

The combined analysis of engineering workload and concurrent license utilization provides important insight into the effectiveness of the proposed approach. Engineering usage records show that software activity recovered following the organization's annual summer vacation period, whereas independent license utilization records indicate that concurrent license consumption remained consistently lower after framework deployment. This observation suggests that the reclaimed licenses primarily originated from applications that remained open but were no longer performing meaningful computational work. Consequently, licenses could be redistributed to active users without interrupting engineering operations.

These findings directly address the research gap identified in Section 2. Existing floating-license management techniques primarily emphasize administrative policies, static timeout mechanisms, or centralized license allocation strategies. While such approaches can improve overall resource management, they typically lack application-level awareness of runtime execution. By incorporating runtime activity monitoring into the license reclamation process, the proposed framework introduces a complementary strategy that enables dynamic and context-aware license optimization without modifying either the commercial CAD application or the enterprise license server.

An additional contribution of the proposed framework is its ability to preserve application state during license reclamation. Rather than terminating inactive sessions or requiring engineers to manually close and reopen applications, the framework suspends all application threads while maintaining the complete execution context. Once a license becomes available, the Application Recovery Module restores execution and allows the CAD application to automatically reacquire a floating license. This approach minimizes workflow disruption and reduces the operational overhead commonly associated with manual license management practices.

7.2 PRACTICAL IMPLICATIONS FOR ENTERPRISE SOFTWARE ASSET MANAGEMENT

Efficient utilization of floating software licenses has become an increasingly important aspect of Software Asset Management (SAM), particularly in industries that depend on high-value engineering applications. Commercial CAD licenses represent a significant operational investment, and organizations frequently encounter situations where concurrent license demand exceeds the available license pool despite a substantial proportion of allocated licenses remaining associated with inactive application sessions.

The proposed framework offers a practical solution that improves the utilization of existing license resources rather than requiring organizations to purchase additional licenses. From an operational perspective, increasing the effective availability of existing licenses may reduce software acquisition costs while maintaining uninterrupted engineering activities. This is particularly valuable for globally distributed engineering organizations where users operate across multiple time zones and where concurrent license demand varies throughout the working day.

Another practical advantage is the framework's non-intrusive deployment model. Because runtime monitoring is implemented entirely on the client workstation, the framework can be integrated into existing enterprise environments without requiring modifications to commercial CAD software, vendor-specific licensing mechanisms, or centralized license servers. This architecture simplifies deployment, reduces implementation risk, and preserves compatibility with vendor-supported

software updates.

The production deployment further demonstrates the scalability of the approach. The framework successfully operated within an enterprise environment comprising approximately 300 virtual workstations supporting approximately 500 engineering users sharing a centralized pool of approximately 265 floating CAD licenses. These deployment characteristics indicate that runtime activity-aware license reclamation is suitable for large-scale industrial environments where license optimization has both technical and economic significance.

7.3 COMPARISON WITH EXISTING LICENSE MANAGEMENT APPROACHES

Existing approaches to floating-license management generally focus on centralized administration, predefined timeout policies, user-driven license return mechanisms, or predictive allocation models. While each of these approaches addresses specific aspects of software asset management, they differ fundamentally from the runtime activity-aware strategy proposed in this study.

Traditional inactivity-based timeout mechanisms typically infer application inactivity from the absence of user input. However, engineering applications often continue performing computational tasks without direct user interaction, including model regeneration, data loading, simulation, or background processing. Conversely, applications may remain open for extended periods while consuming negligible computational resources despite occasional user activity. Consequently, user inactivity alone does not necessarily represent application inactivity.

Administrative license management policies provide another common strategy by enforcing organizational rules regarding license allocation or session duration. Although these approaches can improve fairness across users, they generally lack visibility into the actual execution state of individual applications and therefore cannot distinguish productive computational activity from dormant application sessions.

Recent research has also explored predictive and data-driven techniques for software license optimization, including machine learning methods that forecast license demand based on historical utilization patterns. These approaches primarily support strategic license planning and capacity management rather than real-time operational optimization. In contrast, the proposed framework performs runtime decision-making directly on individual engineering workstations using continuously monitored execution activity. As a result, it complements rather than replaces higher-level software asset management strategies.

The proposed framework therefore occupies a distinct position within the broader landscape of floating-license management by introducing runtime execution awareness as a practical criterion for dynamic license reclamation. This application-centric perspective enables more precise identification of reclaimable licenses while preserving user sessions and requiring no modifications to commercial licensing infrastructure.

7.4 LIMITATIONS AND FUTURE WORK

Although the results demonstrate the practical effectiveness of the proposed framework, several limitations should be acknowledged. First, the evaluation was conducted within a single industrial organization using AVEVA E3D as the target engineering platform. While the underlying methodology is independent of any specific CAD application, additional studies involving different engineering software packages and licensing systems would strengthen the generalizability of the findings.

Second, the current implementation employs a fixed activity threshold of 500 ms accumulated processor execution time within a 900 s monitoring interval. Although this configuration proved effective in the evaluated production environment, optimal threshold values may vary depending on application characteristics, engineering workflows, or hardware configurations. Future research should investigate adaptive threshold selection based on workload characteristics, historical execution patterns, or statistical learning techniques.

Third, the framework evaluates application activity primarily through accumulated processor execution time. While this metric provides a lightweight and platform-supported indication of computational activity, future implementations could incorporate additional runtime indicators such as memory utilization, disk I/O activity, graphics processing workload, or application-specific

telemetry to provide a more comprehensive representation of application state.

Finally, the present study focused on improving floating-license utilization within a single enterprise deployment. Broader evaluations across multiple organizations, industries, and licensing environments would provide further evidence of scalability and applicability. Future work may also investigate integration with enterprise Software Asset Management platforms, adaptive policy management, and predictive resource allocation techniques to further enhance license optimization in large-scale engineering environments.

CONCLUSION

Efficient management of floating software licenses has become increasingly important as engineering organizations seek to maximize the utilization of expensive commercial CAD applications while minimizing operational costs. Conventional license management approaches primarily rely on administrative policies, predefined timeout mechanisms, or user-driven license release, which often fail to distinguish between applications that are actively performing computational work and those that remain open but functionally inactive. This limitation can lead to unnecessary license occupation, reducing the availability of shared software resources for other users.

This paper presented a runtime activity-aware framework for dynamic floating license reclamation that addresses this challenge by continuously monitoring application execution using accumulated processor execution time. The proposed framework operates entirely on the client workstation and requires no modification to either commercial CAD applications or existing license server infrastructure. When an application exhibits negligible computational activity over a configurable monitoring interval, the framework safely suspends its execution, allowing the associated floating license to be automatically reclaimed through the standard license heartbeat timeout mechanism. Subsequently, the Application Recovery Module restores the suspended application, enabling transparent license reacquisition while preserving the complete application state and minimizing disruption to engineering workflows.

The framework was implemented as a Windows-based client application using C# and the Microsoft .NET Framework 4.7 and evaluated in a production enterprise environment supporting approximately 500 engineering users operating across approximately 300 Citrix-hosted virtual workstations with a shared pool of approximately 265 concurrent floating CAD licenses. Unlike many previous studies that rely on laboratory experiments or simulated workloads, the evaluation was based on operational data collected from a real industrial deployment. The assessment combined two independent enterprise datasets: twelve months of engineering software usage records and approximately thirteen months of concurrent floating-license utilization records. The longitudinal evaluation provided a basis for assessing both engineering workload and enterprise license consumption before and after framework deployment.

The experimental results indicate that engineering activity recovered following the organization's annual summer vacation period while concurrent floating-license utilization remained consistently lower after deployment of the proposed framework. These observations are consistent with the intended operation of runtime activity-aware license reclamation, where licenses associated with computationally inactive application sessions become available for redistribution to active users without interrupting ongoing engineering activities. Furthermore, the client-side deployment architecture demonstrated stable operation within an enterprise virtual desktop infrastructure while preserving compatibility with commercial software and existing licensing mechanisms.

From a broader perspective, this work contributes to the field of Software Asset Management by introducing runtime execution awareness as a practical criterion for dynamic floating-license optimization. Rather than replacing existing licensing policies or centralized management systems, the proposed framework complements them by providing application-level intelligence for identifying reclaimable licenses in real time. This combination of runtime monitoring, transparent state preservation, and vendor-independent deployment represents a practical enhancement to existing enterprise license management strategies.

Although the evaluation demonstrates promising results, the study is limited to a single industrial

deployment and a specific commercial CAD platform. Future research will investigate adaptive activity thresholds, support for additional engineering applications and licensing environments, integration with enterprise Software Asset Management platforms, and the application of data-driven techniques to further improve runtime decision-making and license allocation. Extending the framework across multiple organizations and industrial domains will provide additional evidence of its scalability and general applicability.

In summary, the proposed runtime activity-aware framework demonstrates that application execution characteristics can be effectively leveraged to improve floating-license utilization in enterprise engineering environments. By enabling intelligent, non-intrusive, and transparent license reclamation while preserving user productivity, the framework provides a practical and scalable approach to optimizing software resource utilization and offers a promising direction for future research in enterprise software licensing and software asset management.

REFERENCES

[1]. Al-Luhaidan, B. A., Al-Mutairi, F. S., & Al-Khathlan, K. A. (2024). Automated solution for software license compliance. Zenodo. https://doi.org/10.5281/zenodo.14382629

[2]. Alspaugh, T., Scacchi, W., & Asuncion, H. (2010). Software licenses in context: The challenge of heterogeneously licensed systems. Journal of the Association for Information Systems, 11(11), 730–755. https://doi.org/10.17705/1jais.00241

[3]. Camba, J. D., Contero, M., & Company, P. (2016). Parametric CAD modeling: An analysis of strategies for design reusability. Computer-Aided Design, 74, 18–31. https://doi.org/10.1016/j.cad.2016.01.003

[4]. Danglot, B., Monperrus, M., Rudametkin, W., & Baudry, B. (2020). An approach and benchmark to detect behavioral changes of commits in continuous integration. Empirical Software Engineering, 25(4), 2379–2415. https://doi.org/10.1007/s10664-019-09794-7

[5]. David, D. (2023). Web Application for Managing Software Detection Rules in Software Asset Management. Brno University of Technology Digital Library (Brno University of Technology). https://dspace.vut.cz/bitstreams/da1ba48e-3c8d-4d08-9231-67fafc51d822/download

[6]. Doloca, A., & Țănculescu, O. (2011). Floating license management—Automation using web technologies. International Journal of Computers, Communications & Control, 6(4), 615–`

[7]. Gaol, F. L., Santoso, S., & Matsuo, T. (2022). Design and development of the application monitoring the use of server resources for server maintenance. Open Engineering, 12(1), 524–538. https://doi.org/10.1515/eng-2022-0055

[8]. Hasselbring, W., & van Hoorn, A. (2020). Kieker: A monitoring framework for software engineering research. Software Impacts, 5, 100019. https://doi.org/10.1016/j.simpa.2020.100019

[9]. Machal-Fulks, J., & Barnett, C. (2012). Enterprise software licensing: New options—New obligations. Texas Wesleyan Law Review, 18(4), 753–765. https://doi.org/10.37419/TWLR.V18.I4.5

[10]. Papachristou, E., Kyratsis, P., & Bilalis, N. (2019). A comparative study of open-source and licensed CAD systems. Machines, 7(2), 30. https://doi.org/10.3390/machines7020030

[11]. Phillips, T. (2017). Asset Management Software. ScholarWorks@GVSU (Grand Valley State University). https://scholarworks.gvsu.edu/cistechlib/274

[12]. Schlauch, T. (2023). All you need to know about software licenses as a research software engineer. Zenodo. https://doi.org/10.5281/zenodo.7660415

[13]. Wikberg, M. (2010). Software license management from system-integrator viewpoint. Aaltodoc (Aalto University). https://aaltodoc.aalto.fi/handle/123456789/3307

[14]. Wintersgill, N., Stalnaker, T., Otten, D., Heymann, L. A., Chaparro, O., Di Penta, M., German, D. M., & Poshyvanyk, D. (2025). Developers' Perspectives on Software Licensing: Current Practices, Challenges, and Tools. ArXiv.Org. https://arxiv.org/pdf/2510.01096