Improving the Functionality and Performance of a Text Privatization Benchmarking Platform

Ahmet Bilal Akın

01.12.2025, Master Thesis Final Presentation

Chair of Software Engineering for Business Information Systems (sebis)
Department of Computer Science
School of Computation, Information and Technology (CIT)
Technical University of Munich (TUM)
wwwmatthes.in.tum.de

Outline

Motivation – The Privacy-Utility Trade-off

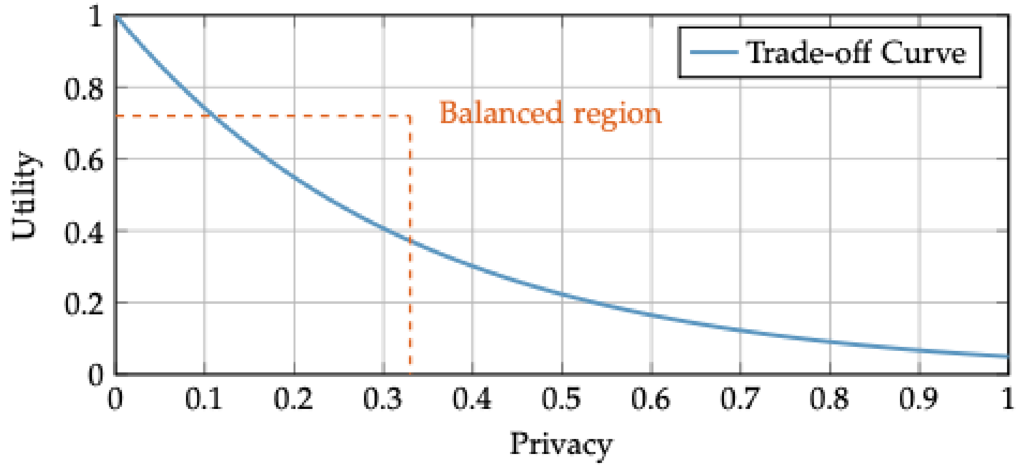


Figure: Illustration of the privacy–utility trade-off.

Source: Adapted from Dwork & Roth (2014).

- **Context:**
  - Digitalization (e.g., healthcare, social media) generates massive amounts of sensitive text data [1].
  - Regulatory compliance, including guidelines such as GDPR, requires the protection of data.
- **The Challenge:**
  - **Privacy:** Masking identifiers (such as names and ages) and quasi-identifiers to prevent re-identification [2].
  - **Utility:** Preserving semantic meaning for downstream NLP tasks.
  - **The Trade-off:** Increasing privacy typically degrades utility; finding the "Balanced Region" is difficult.
- **Research Gap:**
  - Lack of standardized benchmarking platforms that measure *both* dimensions effectively.

[1] Yadav et al., *Indian Dermatology Online Journal* (2023).
[2] Pilán et al., *Applied Soft Computing* (2025).

Problem Statement – Limitations of PrivBench

- **The Starting Point:** An initial modular prototype existed, but faced three scaling barriers :

1. **Performance Bottlenecks:**
    - High latency due to **"Cold Start"** container initialization for every task.
    - No resource reuse; heavy models for some modules are reloaded constantly.
    - Benchmarking requires efficiency to be practical in production [3].
2. **Reproducibility Issues:**
    - No version control for modules; updating an algorithm overwrote previous results.
    - It is not possible to compare results over time (historical data loss).
3. **Usability:**
    - No visibility into task position (queue).
    - Manual entry of metadata is required for every submission.

[3] Liang et al., *HELM: Holistic Evaluation of Language Models* (2022).

Research Questions

RQ1 (Reproducibility & Standardization):

- How can an existing modular evaluation framework for text privatization be extended to enable the standardized, reproducible, and interoperable measurement of privacy, utility, and performance?

RQ2 (Engineering & Performance):

- How can the platform be designed and engineered to fulfill key software quality attributes such as performance, maintainability, extensibility, and fairness in the benchmarking process?

Analysis of the Existing System

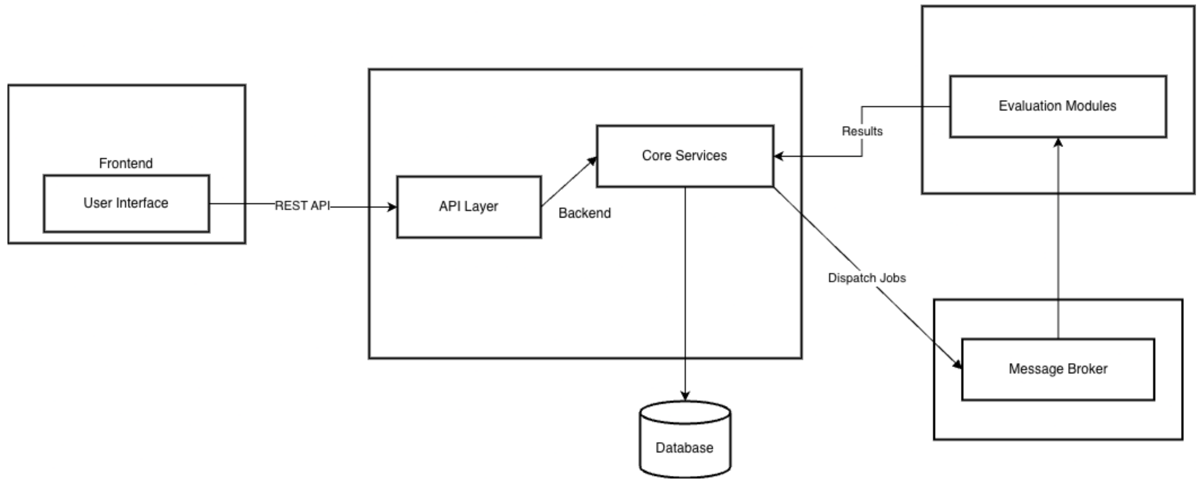


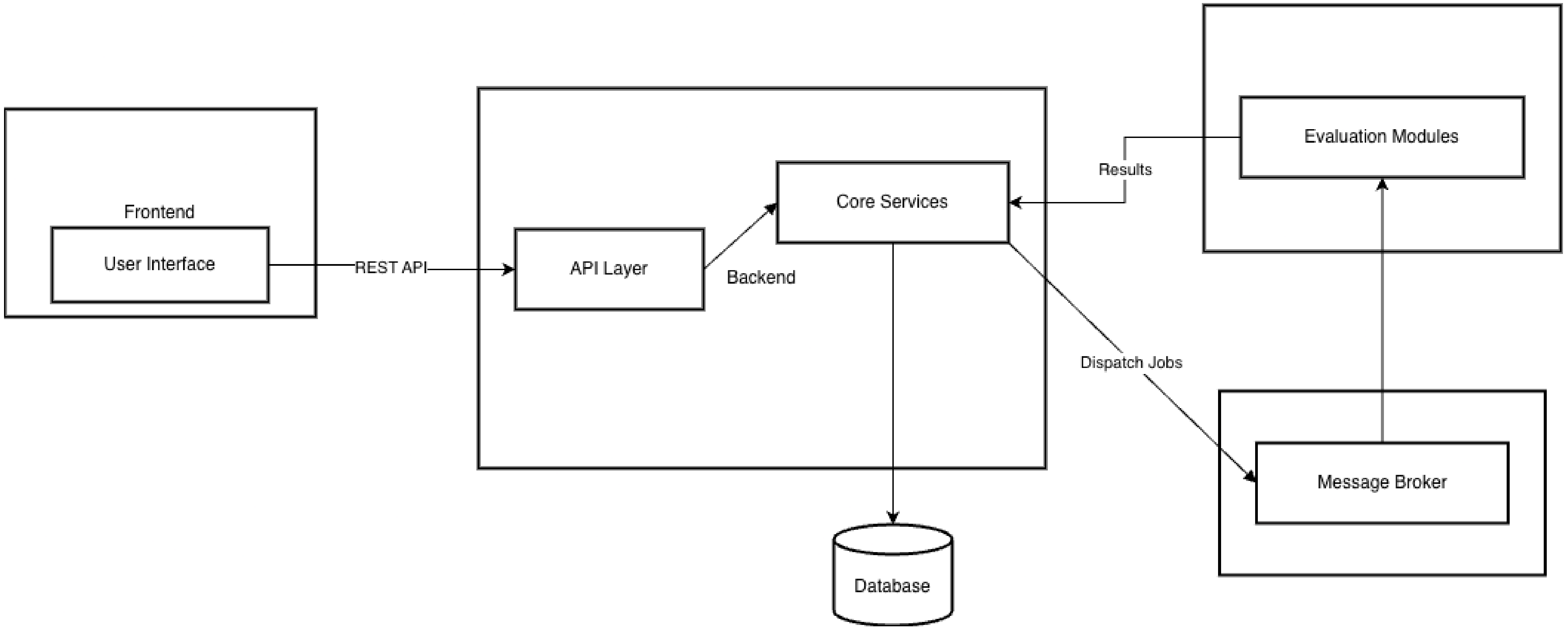
Figure: Component-level architecture of the legacy *PrivBench* system.

Workflow:

1. User uploads submission.
2. System spins up a *new* Docker container.
3. Module runs.
4. The container is destroyed.

Identified Bottlenecks:

- **Operational Complexity:** Repeated container startup/shutdown costs time.
- **Black Box:** No queue visibility for users.
- **Data Traceability (Versioning):** Previous results were overwritten with each module update.


Frontend
User Interface
REST API
API Layer
Backend
Core Services
Database
Results
Dispatch Jobs
Evaluation Modules
Message Broker

Improved System Architecture

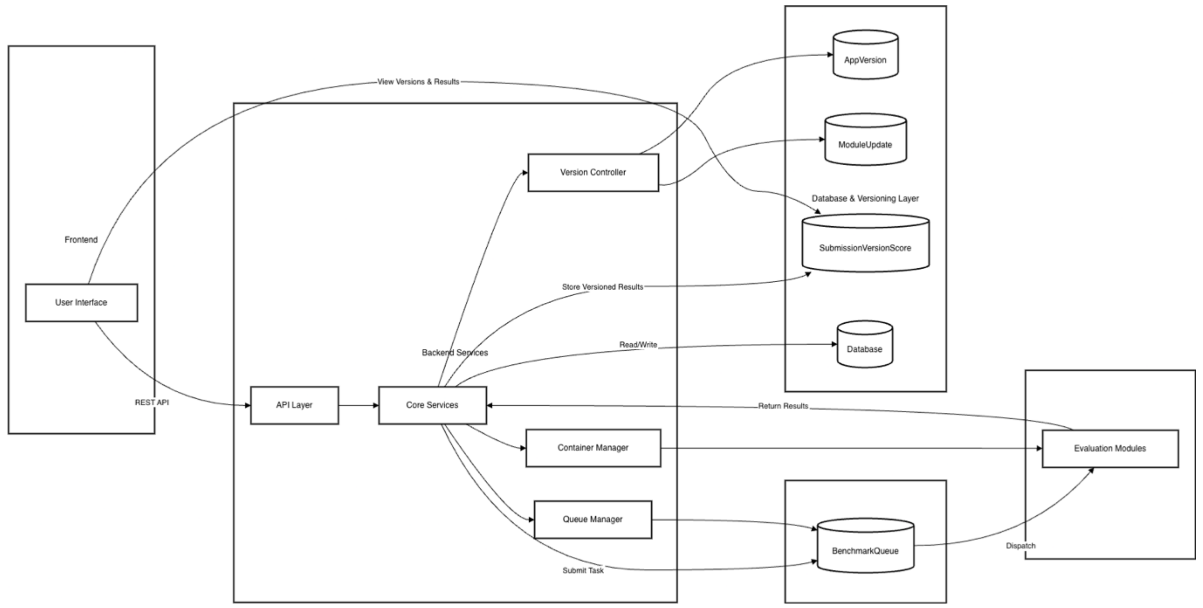
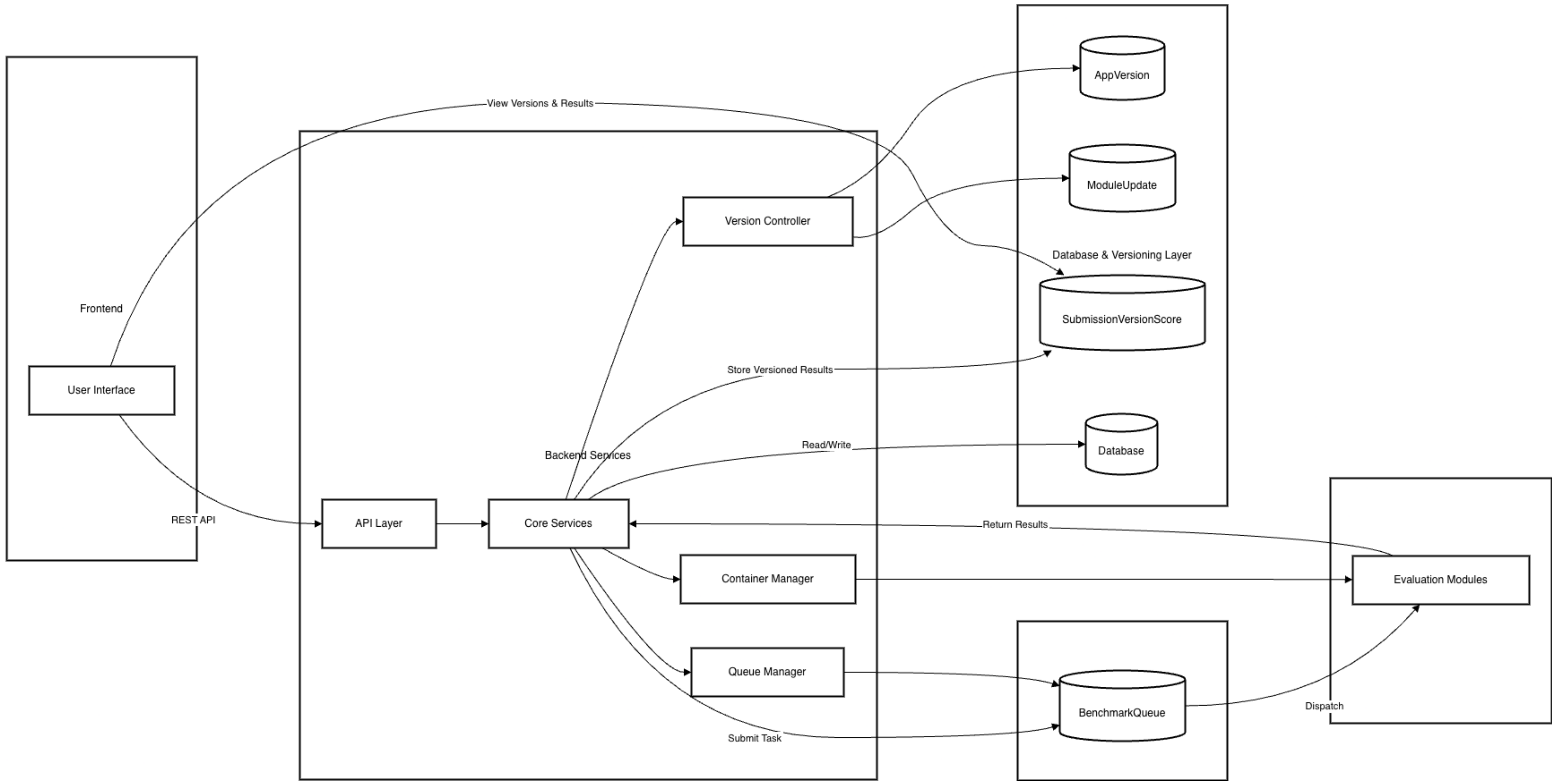


Figure: Component-level architecture of the extended *PrivBench* system.

New Core Components:

- **Container Manager:** Orchestrates Docker lifecycles and resource reuse.
- **Queue Manager:** Implements FCFS scheduling and handles fairness.
- **Version Controller:** Manages app version and module updates.

Database Extension: Added entities for version tracking and templates.


View Versions & Results
Frontend
User Interface
REST API
Backend Services
API Layer
Core Services
Version Controller
Store Versioned Results
Read/Write
Container Manager
Queue Manager
Submit Task
AppVersion
ModuleUpdate
Database & Versioning Layer
SubmissionVersionScore
Database
Return Results
Evaluation Modules
BenchmarkQueue
Dispatch

Performance Optimization (Container Manager)

- **Objective:** Eliminate initialization latency ("Cold Starts").
- **Implementation Strategy:**
  - **Warm Starts:** Containers are initialized *once* at application startup.
  - **Pool Management:** Running containers have an "idle" state with models pre-loaded in memory.
  - **Execution Flow:** Submission -> Assign to Idle Container -> Execute -> Return to Pool.

Benefit and Result: Eliminates initialization latency and model loading overhead.

Scalability & Fairness (Queue Management)

- **Objective:** Prevent server overload and provide user transparency.
- **The Solution:** A benchmark submission queue.
- **Features:**
  - **Serialization:** Validates and enqueues submissions to prevent overload.
  - **Transparency:** Users see "Position: X" and its corresponding status (Waiting, Processing, or Completed).
  - **Fairness:** Ensures First-Come-First-Serve execution.
  - **User Experience:** Enables cancellation of long-running tasks.

Reproducibility (The Versioning System)

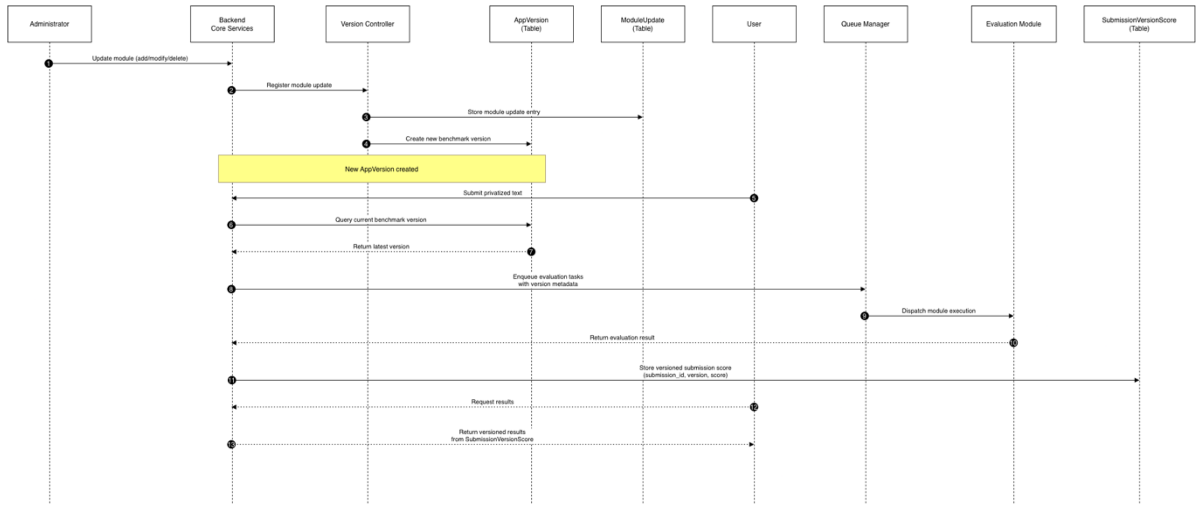

Administrator
Backend Core Services
Version Controller
AppVersion (Table)
ModuleUpdate (Table)
User
Queue Manager
Evaluation Module
SubmissionVersionScore (Table)
1 Update module (add/modify/delete)
2 Register module update
3 Store module update entry
4 Create new benchmark version
New AppVersion created
5 Submit privatized text
6 Query current benchmark version
7 Return latest version
8 Enqueue evaluation tasks with version metadata
9 Dispatch module execution
10 Return evaluation result
11 Store versioned submission score (submission_id, version, score)
12 Request results
13 Return versioned results from SubmissionVersionScore

Figure: Sequence diagram illustrating the versioning workflow.

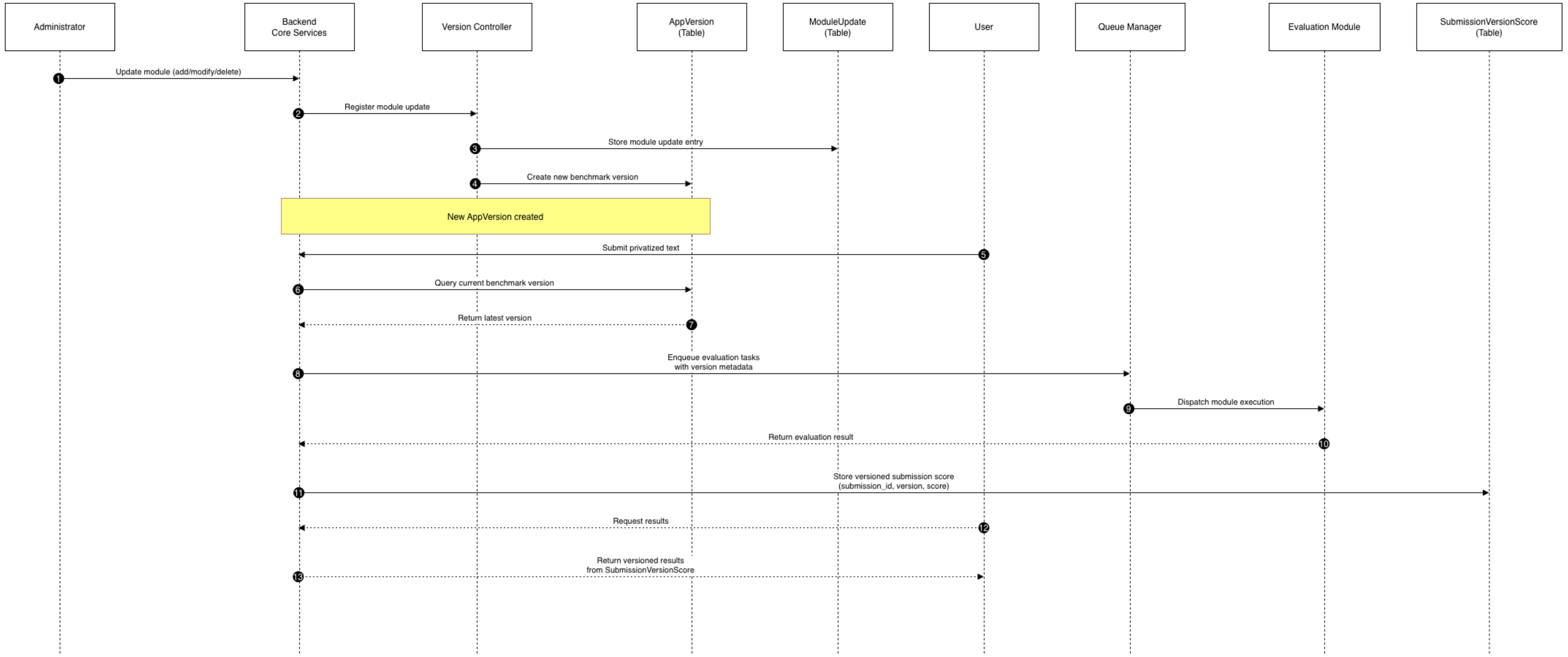
Objective: Ensure results remain valid even as the system evolves.

New Data Models:

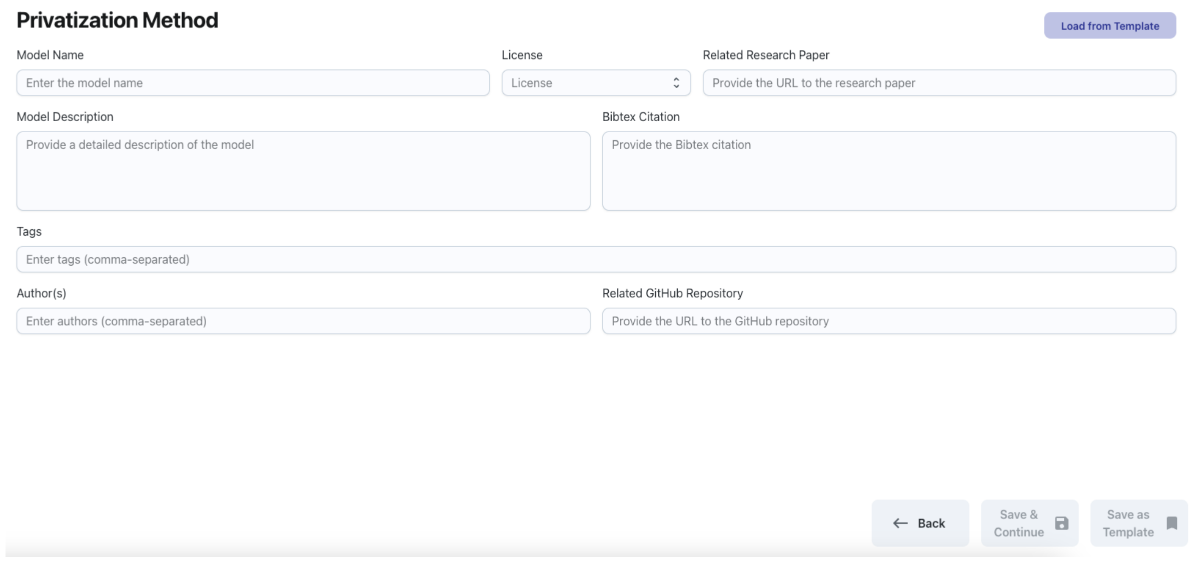
- *AppVersion:* Snapshots of the global system state.
- *ModuleUpdate:* Tracks specific changes (e.g., "v1.1: Updated algorithm").
- *SubmissionVersionScore* links a result to the specific version used during evaluation.

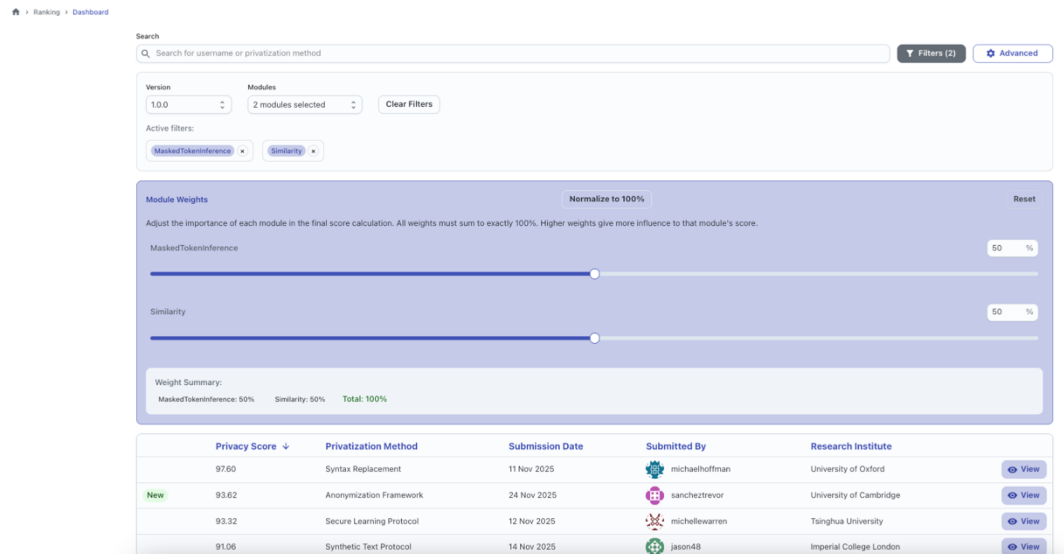
Outcome:

- Enables historical comparison even after system updates.


Administrator
Backend Core Services
Version Controller
AppVersion (Table)
ModuleUpdate (Table)
User
Queue Manager
Evaluation Module
SubmissionVersionScore (Table)
1 Update module (add/modify/delete)
2 Register module update
3 Store module update entry
4 Create new benchmark version
New AppVersion created
5 Submit privatized text
6 Query current benchmark version
7 Return latest version
8 Enqueue evaluation tasks with version metadata
9 Dispatch module execution
10 Return evaluation result
11 Store versioned submission score (submission_id, version, score)
12 Request results
13 Return versioned results from SubmissionVersionScore

Usability Enhancements


Privatization Method
Load from Template
Model Name
Enter the model name
License
License
Related Research Paper
Provide the URL to the research paper
Model Description
Provide a detailed description of the model
Bibtex Citation
Provide the Bibtex citation
Tags
Enter tags (comma-separated)
Author(s)
Enter authors (comma-separated)
Related GitHub Repository
Provide the URL to the GitHub repository
Back
Save & Continue
Save as Template



Adjust the importance of each module in the final score calculation. All weights must sum to exactly 100%. Higher weights give more influence to that module's score.

Weight Summary:
MaskedTokenInference: 50% Similarity: 50% Total: 100%

	Privacy Score ↓	Privatization Method	Submission Date	Submitted By	Research Institute	
	97.60	Syntax Replacement	11 Nov 2025	michaelhoffman	University of Oxford	View
New	93.62	Anonymization Framework	24 Nov 2025	sancheztrevor	University of Cambridge	View
	93.32	Secure Learning Protocol	12 Nov 2025	michellewarren	Tsinghua University	View
	91.06	Synthetic Text Protocol	14 Nov 2025	jason48	Imperial College London	View

Metadata Templates:

- Users define model details (such as authors and citations) once and reuse them.
- Reduces friction for repeated experiments.

Version-Aware Leaderboard:

- Filters rankings by benchmark version.
- Ensures fairness by only comparing scores generated under identical module configurations.

System Demonstration

Workflow to demonstrate:

1. **Submission:** Loading a saved metadata template.
2. **Queuing:** Submitting and pointing out the "Position: 1" status indicator.
3. **Results:** Viewing the final score tagged with "Version 1.1.0".
4. **Comparison:** Filtering the leaderboard to show historical data.

Evaluation Setup

Hardware Environment:

- MacBook Air M1 (8-core CPU, 8GB RAM).
- Represents a standard developer environment.

Workload:

- **Dataset:** Lightweight dataset (100 text rows) to isolate architecture behavior.
- **Repetitions:** 10 independent runs per module type.

Test Modules:

- **Similarity Module:** Lightweight (Transformer-based embeddings).
- **MaskedTokenInference:** Computationally intensive (Masked Language Model).

Performance Results

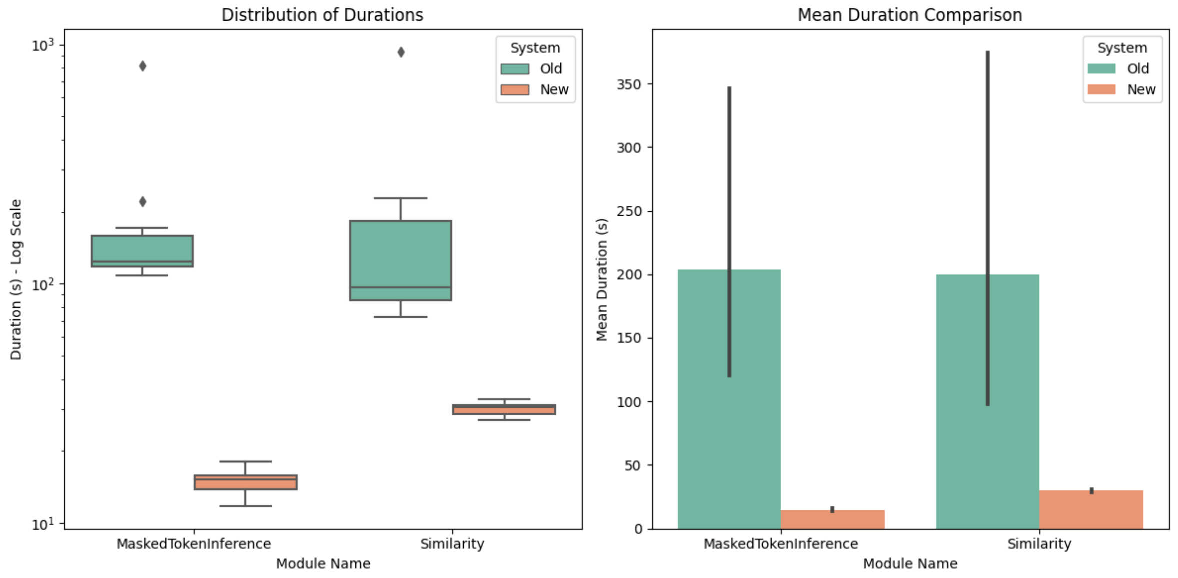


Figure: Performance comparison between the Old and New systems.

Quantitative Findings (Mean Duration): Massive reduction in execution time thanks to warm starts.

- **MaskedTokenInference:** Dropped from **204s** (Old) to **15s** (New).
- **Similarity:** Dropped from **200s** (Old) to **30s** (New).

Speedup: 6x to 13x improvement depending on module complexity.

Conclusion: The Container Manager successfully eliminated the initialization bottleneck.

Usability Results

Transparency: Verified that users can track real-time queue positions.

Efficiency: Template metadata significantly simplified the submission flow.

Clarity: Versioned leaderboards successfully grouped comparable results, solving the data traceability issue.

Limitations & Future Work

1. **Evaluation Environment**
   - *Limitation:* Performance benchmarking was restricted to a local CPU environment.
   - *Future Work:* Conduct full-scale stress testing on the GPU-enabled VM architecture.
2. **Versioning Scope**
   - *Limitation:* System versions module configurations, but not yet the datasets or internal logic files.
   - *Future Work:* Extend the data model to include dataset versioning and module logic tracking for total reproducibility.
3. **Scalability**
   - *Limitation:* The current architecture utilizes a single Celery worker pool.
   - *Future Work:* Implement distributed worker nodes or Kubernetes orchestration for massive parallel loads.

Conclusion

Summary of Contributions:

- **Performance:** Achieved 6x-13x speedup by shifting from Cold Starts to Warm Starts via Container Management.
- **Reproducibility:** Built a Versioning System for standardized comparison.
- **Fairness & Usability:** Implemented Queue Management to handle concurrent submissions.
- **Final Status:** Transformed PrivBench from a prototype to a production-ready platform.

Q&A

Thank you for your attention.

References

1. **[Yadav et al., 2023]** Data Privacy in Healthcare: In the Era of Artificial Intelligence. *Indian Dermatology Online Journal*.
2. **[Pilán et al., 2025]** Truthful text sanitization guided by inference attacks. *Applied Soft Computing*.
3. **[Dwork & Roth, 2014]** The Algorithmic Foundations of Differential Privacy.
4. **[Liang et al., 2022]** HELM: Holistic Evaluation of Language Models.

B. Sc.

Ahmet Bilal Akın

bilal.akn@tum.de

Technical University of Munich (TUM)
TUM School of CIT
Department of Computer Science (CS)
Chair of Software Engineering for Business Information Systems (sebis)

Boltzmannstraße 3
85748 Garching bei München

+49.89.289.

wwwmatthes.in.tum.de