Designing a Hybrid ORM Architecture for High-Performance .NET Applications

Mehrdad Azimi
Independent Software Developer
AzimiDeveloper@gmail.com

MohammadReza Moumivand
Independent Software Developer
Mohamadrezamvd@gmail.com

Abstract

Context
Modern .NET applications frequently face a performance-versus-maintainability trade-off when selecting object-relational mapping (ORM) frameworks. Full-featured ORMs like Entity Framework Core (EF Core) offer rich abstraction and integration with LINQ and change tracking, but often underperform in high-throughput environments. Conversely, micro-ORMs such as RepoDb or Dapper deliver superior raw performance but lack features necessary for domain modeling, maintainability, and developer ergonomics. This dichotomy presents a challenge for building scalable, enterprise-grade systems.

Objectives
This research aims to design, implement, and evaluate a hybrid ORM architecture for .NET that combines EF Core and RepoDb in a runtime-adaptive pattern. The objective is to provide seamless performance optimization for high-load transactional systems, while maintaining high-level abstractions for complex business queries, thus removing the need for exclusive ORM commitment in enterprise systems.

Methods
We introduce a hybrid architecture centered around a RepositoryResolver<T> interface that dynamically delegates database operations to either EF Core or RepoDb, based on criteria such as operation type, data size, and query complexity. The system is implemented in C# targeting .NET 6 and .NET 8, utilizing Dependency Injection for component management. Performance benchmarks were conducted using BenchmarkDotNet and PostgreSQL on a dataset of 10,000 records. Furthermore, a production-grade financial logging service was used to validate real-world applicability.

Results
Benchmarking showed that the hybrid architecture matched RepoDb's performance in bulk insert (940 ms) and single lookup (4 ms), while EF Core alone took 3250 ms and

18 ms respectively. In a production deployment for logging over 1,000 financial transactions per second, the architecture reduced peak CPU usage by 38% and query response times by 65%. Importantly, the hybrid maintained compatibility with EF Core's navigation properties and LINQ, preserving developer productivity.

Conclusion
The proposed hybrid ORM design enables .NET systems to exploit the strengths of both EF Core and RepoDb, achieving high-throughput efficiency without sacrificing maintainability. This contribution is valuable for software engineers and researchers who design data access layers for enterprise-scale applications. The architecture is extensible and supports adaptation to varied domains such as financial services, analytics platforms, and auditing systems.

---

1. Introduction

ORM frameworks simplify database interaction in object-oriented applications. EF Core, the primary ORM in the .NET ecosystem, enables developers to use LINQ, manage migrations, and track state with minimal boilerplate. However, these conveniences come at a performance cost, especially in scenarios involving large-scale data ingestion or real-time processing.

On the other end, lightweight ORMs like **Dapper** and **RepoDb** offer raw speed by generating SQL directly and bypassing unnecessary abstraction layers. Yet, their use demands manual query management and offers minimal support for complex entity relationships, change tracking, and domain-driven design (DDD) principles.

In real-world enterprise environments, applications often require both:

- The **speed** of micro-ORMs for hot paths like logging or telemetry
- The **abstraction** of full-featured ORMs for business logic, auditing, and reporting

This paper introduces a **hybrid ORM model** that unifies both paradigms through runtime polymorphism and dependency injection. The system routes requests dynamically based on operation type and payload characteristics.

Key contributions:

- A plug-and-play hybrid ORM system for .NET 6+
- Quantitative benchmarking of EF Core, RepoDb, and the hybrid design
- Production use-case implementation
- Practical guidelines for architectural adoption

2. Related Work

Previous works have studied ORM performance in .NET ecosystems.

- **Smith et al. (2023)** showed that Dapper outperforms EF Core by 3–5× in bulk operations but requires hand-written SQL for each use case [1].
- **Zhang and Lee (2022)** discussed trade-offs in ORM design, noting that enterprise systems often need to balance abstraction and control [2].
- **Anderson (2021)** described microservice architectures that combine NoSQL and SQL stores, but did not explore hybrid ORM strategies [3].
- **RepoDb documentation** claims 2–4× speedups over EF Core [4], but no peer-reviewed evaluation exists.
- In Java ecosystems, hybrid persistence is studied (e.g., Hibernate + JOOQ), but no established equivalent exists in .NET.

Our approach is the first peer-reviewed implementation of **runtime-adaptive ORM switching in .NET**, based on real-world constraints.

3. Hybrid ORM Architecture

3.1. Architectural Overview

The architecture introduces a **RepositoryResolver<T>** interface backed by two concrete implementations:

- EfRepository<T> using DbContext
- RepoDbRepository<T> using direct SQL with IDbConnection

These repositories are registered via dependency injection. At runtime, the resolver chooses based on:

- Operation type (Insert, Update, Query)
- Expected volume
- Query complexity (e.g., LINQ expression tree)

3.2. Routing Logic

```
public async Task<IEnumerable<T>> QueryAsync(Expression<Func<T, bool>>
predicate)
```

```
{
    if (IsSimplePredicate(predicate) && EstimateResultCount(predicate) < 1000)
        return await _repoDb.QueryAsync(predicate);

    return await _efCore.QueryAsync(predicate);
}
```

3.3. Dynamic Injection

.NET's built-in DI container is used to register each repository as:

services.AddScoped<IRepository<T>, RepositoryResolver<T>>();

---

4. Implementation and Design

4.1 Class Structure

- BaseEfRepository<T>: Wraps DbContext.Set<T>(), provides standard CRUD.
- BaseRepoDbRepository<T>: Uses RepoDb's QueryAll<T>(), InsertAll<T>(), etc.
- RepositoryResolver<T>: Runtime strategy switcher.

4.2. Configuration Options

- HybridOrmOptions allow developers to specify routing rules per model or operation.
- Auto fallback is enabled if RepoDb fails (e.g., joins or navigation properties).

---

5. Benchmarking and Performance Evaluation

5.1. Experimental Setup

- **Hardware**: Intel Core i7, 16GB RAM, SSD
- **DB**: PostgreSQL 13.3
- **Rows**: 10,000 inserts, 10,000 lookups
- **Framework**: .NET 8.0
- **Tools**: Stopwatch, MemoryProfiler, BenchmarkDotNet

5.2. Results

Operation	EF Core	RepoDb	Hybrid
Bulk Insert (10k)	3,250 ms	940 ms	940 ms
Single Lookup	18 ms	4 ms	4 ms
One-to-Many Join	780 ms	620 ms	620 ms

Figure 1 – Hybrid ORM matches RepoDb for performance-critical operations, while preserving EF Core ergonomics.

---

6. Case Study: Financial Transaction Logging

A real-world implementation in a financial payment gateway demonstrated the architecture.

Scenario:

- 1,000 transactions/sec
- High-volume insert logs with minimal delay
- Role-based access reporting with joins and filters

Hybrid ORM Use:

- RepoDb handles **real-time logs** (AuditLog table)
- EF Core handles **reporting queries**, joins with metadata

Results:

Metric	EF Core Only	Hybrid
Peak CPU Load	78%	48%
Report Query Time	460 ms	160 ms
Log Insert Latency	65 ms	18 ms

---

7. Comparative Analysis of ORM Tools

7.1 Feature Comparison Table

Feature	EF Core	Dapper	RepoDb	Hybrid

LINQ	☑	✗	⚠	☑
Performance (Bulk Ops)	✗	☑	☑	☑
Abstraction	☑	✗	⚠	☑
Migrations	☑	✗	✗	☑
Joins / Navigation	☑	✗	⚠	☑

Figure 2 – Hybrid ORM inherits RepoDb's speed + EF Core's flexibility.

---

8. Discussion

8.1 Trade-offs

- EF Core is ideal for rich domains and developer productivity
- RepoDb excels in raw speed and control
- Hybrid ORM introduces **runtime complexity**, but contains it within 1 class (~200 LOC)

8.2 Limitations

- Heuristic-based resolution might misroute rare cases
- Devs must still understand both tools to debug edge scenarios

---

9. Conclusion and Future Work

This paper proposes and evaluates a **Hybrid ORM** design for .NET that reconciles abstraction and speed. It:

- Achieves RepoDb-like performance on critical paths
- Maintains EF Core integration for complex domains
- Reduces infrastructure load in production
- Requires minimal effort to integrate into existing projects

Future work:

- ML-based repository routing
- Visual Studio tooling for ORM hints

- Integration with NoSQL backends (e.g., MongoDb)

References

1. Smith J., Patel R., "Performance Evaluation of ORMs in .NET", IEEE TSE, 2023
2. Zhang H., Lee S., "Architectural Trade-offs in ORM Frameworks", ACM Soft. Arch., 2022
3. Anderson P., "Hybrid Persistence Strategies", J. Ent. Arch., 2021
4. RepoDb Docs,
5. Microsoft Docs - EF Core
6. Nguyen H., "Dynamic Routing in Middleware", ACM SIGSOFT, 2022
7. Williams L., "Hybrid ORM in Practice", Springer, 2023
8. O’Connor M., "Micro-ORM Patterns", J. Database Mgmt., 2020
9. Roberts T., "Bulk Data Optimizations", J. Software Tools, 2021
10. Li F., "ORM Adaptation in Enterprise", ACM Trans. DB Sys., 2021