

Course Title: Research Methodology in Computer Science

Course Code: AI800

Weighting: 20%

Lecturer: Dr. Mohammed Anbar

Hand-out Date: 20 June 2026

Hand-in Date: 04 July 2026

Hand-in Method: Online Submission via E-Learning.

Research Title: Architectural Strategies for Integrating Machine Learning into Enterprise .NET Ecosystems.

Table of contents:

- Summary of each research paper.
- Critical Analysis.
- Connections.
- Conflicts between the findings.
- Synthesis Matrix for Literature Review.
- References.

Summary of each research paper:

Paper 1: Machine Learning at Microsoft with ML.NET.

This paper focuses on overcoming the engineering disconnect that usually exists between data science prototyping environments and production software deployment. In a typical enterprise architecture, machine learning workflows are frequently decoupled from standard development process which results in very insufficient, custom container deployments and costly code rewrites when moving a model into production applications. The author proposes ML.NET to solve this integration issue. The framework relies on a DataView structure, which borrows ideas from relational databases to process data streams efficiently. Rather than running tasks step-by-step, it organizes transformations into Directed Acyclic Graphs (DAGs) and uses lazy evaluation. This allows the runtime to compile the whole pipeline into a single JIT-compiled execution. A major advantage of this design is its memory management, by using pull-based cursors, the system streams data directly from disk. This means developers can process massive datasets without worrying about hitting system RAM limits. The authors validated the framework by running multicore scale up and sample-size benchmarks against prominent open-source toolkits, including Scikit-learn and H2O. The performance evaluation was conducted on azure virtual machine using three large scale production workloads: the Criteo dataset (~45M records), the Amazon Review dataset (~17M records) and the Flight Delay dataset (~1M records). Benchmark testing on Azure showed that ML.NET processes massive datasets much faster and more accurately than H2O or Scikit-learn across large scale out-of-memory enterprise scenarios. However, the study shows a distinct methodological limitation during small sample execution. When trained on a restricted subset (0.1%) of the Flight Delay data, the framework's LightGBM trainer overfits the data, which causes higher test errors than simpler Scikit-learn models. Finally, the technical analysis reveals that using Python DataFrame bindings (NimbusML-DF) to bridge the frameworks introduces a severe read/write latency barrier, as the memory overhead required to convert data frames into streaming DataViews completely dominates the runtime performance.

Paper 2: From the Edge to the Cloud: Model Serving in ML.NET.

The authors tackle the high memory overhead and latency issues that come with deploying ML models in standard, black-box containers. Traditional containers are often too heavy and inflexible for serving models on resource-constrained hardware like IoT edge devices. To fix this, the authors have built PRETZEL, a 'white-box' runtime that treats ML pipeline execution more like a database query optimization problem. Instead of running opaque code, PRETZEL breaks models down into Directed Acyclic Graphs (DAGs). It translates the ML.NET pipeline into an intermediate language called Flour, and then uses a compiler named Oven to apply standard database optimizations like stage fusing. This approach cuts out the JIT compilation overhead and strips away the unnecessary memory buffers ML.NET normally uses. However, the authors admit a major limitation: the data science field lacks standard software engineering practices, so things like rigorous unit testing and lifecycle management aren't structurally supported for these highly optimized runtimes yet.

Paper 3: Integrating Machine Learning with Fullstack Development Using ML.NET.

The authors look at how to solve the problem of standard, static web applications lacking real-time predictive features. The authors prototyped multi-tier architecture using an ASP.NET Core backend to handle the data processing and ML training natively. Using ML.NET, the backend runs algorithms like SDCA for regression and uses AutoML to handle hyperparameter tuning automatically. Instead of a separate data science stack, these predictions are pushed straight via REST APIs to standard React or Angular frontends. The authors found that this setup makes it much easier for regular software engineers to add real-time ML features, like dashboards or recommendation engines, into standard web endpoints. However, they point out some major drawbacks. Pushing large datasets across these web tiers causes noticeable latency and state-management issues. Furthermore, if the models touch sensitive data, they still need heavily isolated containers and highly secure APIs to remain compliant.

Paper 4: Integrating Cloud Services for Distributed AI Model Training in .NET.

The authors for this paper tackle the performance bottlenecks of training massive ML models on a single machine. Their goal was to build a cost-effective, horizontally scalable distributed training pipeline directly within .NET. To solve this, the authors built a distributed training framework using .NET 6 and ML.NET on Azure. They containerized the training logic with Docker and used Azure Kubernetes Service (AKS) to handle the load balancing and auto-scaling. They tested the architecture by deploying a recommendation engine across GPU-equipped nodes, pulling data straight from Azure Blob Storage. The results showed that the AKS deployment cut training time in half while maintaining 85% accuracy, and the auto-scaling kept cloud costs down. However, distributed environments come with heavy infrastructure trade-offs. The authors noted that syncing parameters across multiple nodes creates massive communication latency. Furthermore, distributing workloads across cloud nodes exposes the data to privacy risks, making strict RBAC and encryption mandatory for compliance.

Paper 5: AIRMan: An Artificial Intelligence (AI) Risk Management System.

This paper addresses the lack of practical implementation guidelines in new regulations like the EU AI Act. Their goal was to build a concrete risk management architecture for governing high-risk AI. To do this, they mapped existing security standards, specifically ISO 31000, ISO/IEC 27005, and COBIT 5 directly onto the legal requirements of the EU Act. The result is the AIRMan framework, built on a standard Plan-Do-Check-Act (PDCA) lifecycle. It maps AI threats from sources like MITRE ATLAS across five operational layers of an enterprise, from high-level business goals down to IT infrastructure. The study concludes that AI risk cannot just be left to developers, it requires continuous monitoring across the whole organization to maintain compliance. However, they noted a major limitation regarding AI explainability. External auditors cannot easily verify if security controls are working without costly model retraining. Also, because ML models constantly adapt, standard cybersecurity tests like fuzzing are useless and can introduce permanent bias into the system.

Paper 6: Responsible AI Data Architecture: Embedding GDPR and PII Compliance into MLOps Pipelines at Enterprise Scale.

The author looks at the clash between fast-paced MLOps and strict data privacy laws like GDPR. The goal of this paper was to design an enterprise-scale architecture that bakes PII compliance directly into machine learning data pipelines. To achieve this, the author designed a Responsible AI Data Architecture that applies privacy control across all seven stages of the MLOps lifecycle. Instead of relying on manual checks, the framework uses Policy-as-Code gates directly inside the CI/CD pipelines. The author tested this using real-world case studies, integrating tools like automated NLP for discovering PII, secure tokenization vaults, and Merkle trees for tamper-evident audit trails. The findings prove that automating compliance inside DevOps pipelines eliminates manual audit errors and safely sanitizes data during inference without causing major delays. However, the author points out significant engineering trade-offs: running automated NLP for PII classification requires heavy compute power, and using customer-managed encryption keys creates a single point of failure that could permanently lock out data access if the keys are lost.

Critical Analysis:

Evaluation of Methodology and Data Sources:

The reviewed papers use a mix of real-world prototyping and standard benchmarks. Paper 1 relies on the massive Criteo and Amazon datasets to test streaming, which is great for scale, but the methodology fails on small samples, causing overfitting. Papers 3 and 4 take a more practical route by prototyping distributed architectures directly in Azure. Instead of raw data, Papers 2, 5, and 6 evaluate frameworks against established standards like ISO/IEC 27005 and COBIT 5. This is solid for governance, but it means their findings are more qualitative than quantitative.

Core Strengths of the Proposed Architectures:

The biggest advantage across these papers is getting ML out of isolated data science environments and into standard .NET production. By treating ML pipelines as DAGs and running them natively, these architectures drastically cut down on latency and memory overhead. Furthermore, Papers 4, 5, and 6 provide excellent blueprints for enterprise scaling. Using AKS for autoscaling and tying CI/CD pipelines directly to compliance frameworks makes it much easier to deploy AI without violating the EU AI Act or GDPR.

Critical Weaknesses and Limitations:

The main flaw across the research is the heavy operational tradeoffs. Paper 1 shows massive latency when trying to bridge ML.NET with Python. As things scale up, distributed setups like the one in Paper 4 create huge communication overhead when syncing nodes. On the governance side, standard security audits do not work on these models. As Paper 5 points out, standard cybersecurity fuzzing just introduces bias into the ML system. Finally, Paper 6 shows that automating compliance requires massive compute power, and relying on customer managed keys creates a single point of failure that could lock the enterprise out of its own data.

Connections:

There is a clear consensus across Papers 1, 2, and 3 that native ecosystem integration is the best path for enterprise AI. All three studies agree that treating ML models as standard software components-rather than isolated, black-box containers-drastically improves execution speed. By keeping the pipeline within .NET, developers avoid the heavy memory overhead usually associated with external data science deployments. On the enterprise side, Papers 4, 5, and 6 closely align on infrastructure. They all highlight that scaling these models requires tight integration with cloud-native services (like Azure AKS) and that regulatory compliance must be automated directly into the architecture.

Conflicts between the findings:

Despite these connections, there is a massive conflict between system performance and IT governance. Papers 2 and 3 push for rapid, highly optimized model deployments to support real-time web applications. In direct contrast, Papers 5 and 6 prove that this need for speed clashes heavily with compliance. The fast, agile deployments praised in Paper 3 are severely bottlenecked by the heavy compute power needed for the automated PII scanning demanded in Paper 6.

There is also a direct technical conflict regarding scalability. Paper 1 focuses on optimizing single-machine setups using data streaming. Paper 4 completely rejects this approach for large models, arguing that single nodes are a bottleneck and workloads must be horizontally distributed. However, moving to the distributed architecture in Paper 4 introduces massive node-sync latency, which directly undermines the exact speed optimizations that Papers 1 and 2 worked so hard to achieve.

Synthesis Matrix for Literature Review

Paper	Authors	Year	Research Design	Key Findings/Results	Methodology	Limitations
Machine Learning at Microsoft with ML.NET.	Zeeshan Ahmed, Saeed Amizadeh, Mikhail Bilenko, et al.	2019	Conducted scale-out and scale-up experiments comparing ML.NET against Scikit-learn and H2O across 3 datasets (Criteo, Amazon Review, Flight Delay).	ML.NET outperforms Scikit-learn and H2O in speed and accuracy for most cases, efficiently handling large data via lazy evaluation and pull-based streaming.	Outlined ML.NET architecture centered around the DataView abstraction and cursoring.	Python-based libraries suffer from global interpreter locks; in small-sample tests, ML.NET's LightGBM overfits and underperforms Scikit-learn.
From the Edge to the Cloud: Model Serving in ML.NET.	Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Markus Weimer, Matteo Interlandi	2018	Re-engineered model serving by casting ML pipeline execution as a database multi-query optimization problem.	White-box serving (PRETZEL) optimizes performance via operator reordering, resource sharing, and eliminating reflection/JIT compilation.	Architected PRETZEL, using a compiler (Oven) for Ahead-Of-Time (AOT) compilation of DAGs.	Standard containerized serving introduces network overhead and memory bloating; ML.NET's default execution ignores holistic pipeline optimizations.
AIRMan: An Artificial Intelligence (AI) Risk Management System	Simon Tjoa, Peter Kieseberg, Marlies Temper, Jakob Zanol, Markus Wagner, Andreas Holzinger	2022	Analytical review of existing governance frameworks and the EU AI Act proposal to propose AIRMan.	Proposed AIRMan to systematically identify, assess, treat, and monitor AI risks across different enterprise layers.	Structured PDCA (Plan-Do-Check-Act) lifecycle aligned with ISO Annex SL.	The "explainability problem" makes auditing security countermeasures nearly impossible without highly costly re-training; continuous model adaptation complicates standard security checks.
Integrating Machine Learning with Fullstack Development Using ML.NET	Thiyagarajan Mani Chettier, Venkata Ashok Kumar Boyina	2025	Applied via full-stack case studies (e-commerce, predictive dashboards, sentiment analysis).	ML.NET simplifies integration via robust AutoML and APIs, providing scalable predictive intelligence without deep data science knowledge.	Systematic integration methodology bridging backend (ML.NET, AutoML, SQL Server) and frontend (Angular/React).	Dealing with massive datasets, optimizing model performance, and ensuring data privacy/regulatory compliance require additional rigorous handling.
Responsible AI Data Architecture: Embedding GDPR and PII Compliance into MLOps Pipelines at Enterprise Scale	Narendra Mangala	2026	Case studies (automotive clickstream analysis, telecommunications chatbot).	Compliance enforced as "Policy-as-Code"; PII shields intercept/sanitize input during real-time inference without critical latency overhead.	Formulated an architectural design mapping MLOps lifecycle stages to privacy/security controls.	Encryption introduces latency; ML-based PII detection requires intense computational resources and annotated training data; complete automated data tagging accuracy remains challenging.
Integrating Cloud Services for Distributed AI Model Training in .NET	Emma Oye, Edwin Frank, Jane Owen	2024	Practical case study developing a collaborative filtering recommendation system.	Reduced training time by ~50%; achieved 85% model accuracy; maintained cost efficiency via auto-scaling.	Modular framework using Azure, .NET 6, ML.NET, Docker, and Azure Kubernetes Service (AKS).	Data synchronization complexities, high communication overhead, latency and bandwidth constraints, strict data privacy/security challenges.

References:

- Ahmed, Z., Amizadeh, S., Bilenko, M., Carr, R., Chin, W.-S., Dekel, Y., Dupre, X., Eksarevskiy, V., Filipi, S., Finley, T., Goswami, A., Hoover, M., Inglis, S., Interlandi, M., Kazmi, N., Krivosheev, G., Luferenko, P., Matantsev, I., Matusevych, S., ... Zhu, Y. (2019). Machine Learning at Microsoft with ML.NET. The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), 1-11. https://dl.acm.org/doi/abs/10.1145/3292500.3330667?_cf_chl_f_tk=bcJE3grU8NTbdk1g.o5EuMvDbGYIT1zoAEtGsOsbUY8-1783170791-1.0.1.1-eY_NeeKjSthkTN6tlzVSmeAr3LfM9PtIplisVWnu73Y
- Chettier, T. M., & Boyina, V. A. K. (2025). Integrating Machine Learning with Fullstack Development Using ML.NET. International Journal of Computer Sciences and Engineering, 13(1), 17-23. https://www.researchgate.net/profile/Thiyagarajan-Mani-Chettier/publication/390300770_Integrating_Machine_Learning_with_Fullstack_Development_Using_MLNET/links/67e8136695231d5ba59f3ad1/Integrating-Machine-Learning-with-Fullstack-Development-Using-MLNET.pdf
- Lee, Y., Scolari, A., Chun, B.-G., Santambrogio, M. D., Weimer, M., & Interlandi, M. (2018). PRETZEL: Opening the black box of Machine Learning Prediction Serving. OSDI, 611-626. <https://spl.snu.ac.kr/assets/paper/p46.pdf>
- Mangala, N. (2026). Responsible AI Data Architecture: Embedding GDPR and PII Compliance into MLOps Pipelines at Enterprise Scale. Canadian Journal of Marketing Research, 16(1), 107-124. <https://canadian-jmr.com/index.php/cjmr/article/view/132>
- Oye, E., Frank, E., & Owen, J. (2024). Integrating Cloud Services for Distributed AI Model Training in .NET. https://www.researchgate.net/profile/Micheal-Wilson-2/publication/387298391_Integrating_Cloud_Services_for_Distributed_AI_Model_Training_in_NET/links/6766d8be00aa3770e0b29651/Integrating-Cloud-Services-for-Distributed-AI-Model-Training-in-NET.pdf
- Tjoa, S., Kieseberg, P., Temper, M., Zanol, J., Wagner, M., & Holzinger, A. (2022). AIRMan: An Artificial Intelligence (AI) Risk Management System. <https://ieeexplore.ieee.org/abstract/document/10145175>