Research Project Exhibition 2026

M.Sc. in Software Solutions Architecture, and the M.Sc. in DevOps,

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Technology Ireland ICT Skillnet

FOREWORD

Today's Research Project Exhibition is an occasion to celebrate the outstanding achievements of our taught Master's students from the M.Sc. in Software Solutions Architecture and the M.Sc. in DevOps programmes. The students presenting their research projects today began their Master's journeys in January 2024. While it may feel like only yesterday, a great deal has been accomplished in that time.

As the School of Enterprise Computing and Digital Transformation, we are delighted to take this opportunity to express how proud we are of you as graduates. Your dissertation work, situated at the cutting edge of technology and closely aligned with contemporary industry practice, represents some of the strongest work produced by students within this School. As a relatively new School, we are actively engaged in an extensive consultation process around our mission, vision and activities, and we warmly welcome comments, feedback and input on how our work is currently delivered and how it may continue to evolve.

While these programmes are taught here at TU Dublin, they are inherently collaborative in nature. They have been developed in close partnership with industry, with programme needs refined and proposed by Technology Ireland Skillnet and their ICT employer partners, who identified a clear market demand. TU Dublin responded by working collaboratively to translate industry-defined requirements into academically rigorous Master's programmes.

For the Software Solutions Architecture programme, we are proud to partner with the International Association of Software Architects (IASA) and their Irish partners, the Irish Computer Society (ICS). This programme is formally recognised by IASA and supports eligibility for membership of the association.

Today also marks an important point in the academic cycle, with a new cohort commencing their studies and another cohort progressing into the final stage of their programmes. While they still have much to learn, they too will soon reach this milestone. The creation, sharing and transmission of knowledge lies at the heart of a university's mission, and delivering this within fast-moving technological fields that are directly relevant to industry is central to the role of a Technological University.

We hope you enjoy the project symposium and take pride in the achievements being celebrated today.

Finbarr Feeney PhD / Head of School
School of Enterprise Computing and Digital Transhformation
OT Bhaile Átha Cliath / TU Dublin - Tallaght Campus
D24 FKT9
Ireland

M.Sc. in Software Solutions Architecture, and the M.Sc. in DevOps

RESEARCH PROJECT SYMPOSIUM AGENDA

23rd January 2026

14:00 - 14:30 Opening Addresses

Gary Clynch, School of Enterprise Computing and Digital Transformation, TU Dublin
Jimmy Doody, Head of Discipline, School of Enterprise Computing and Digital Transformation, TU Dublin
Sean McHugh, Head of Discipline, School of Enterprise Computing and Digital Transformation, TU Dublin
Dr. Barry Feeney, Head of School of Enterprise Computing and Digital Transformation, TU Dublin

14:30 - 15:30 Speaker - Imelda Casey, National Cyber Security Centre

Strengthening Ireland's Cybersecurity and Resilience

15:30 - 17:30 Poster Presentations

Technology Ireland ICT Skillnet

TU DUBLIN
OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

Online MSc in DevOps

With most technology organisations moving their delivery platforms to a DevOps approach the shortage of people with cross sectional skills in DevOps is now acute. Developed by industry as a direct response to this need this first-ever Master's degree in DevOps aims to fill these important talent gaps and give credit, recognition and credibility to technologists working in this field.

The advantages of Development Teams and Operations Teams collaborating to improve the delivery of technology solutions has meant a rapid adoption of DevOps approaches to the Software Development Lifecycle. Closely associated with Lean and Agile concepts in enhancing the delivery of technology solutions, the DevOps approach has impacted very rapidly on the Technology industry.

Most existing DevOps 'specialists' grow or develop into their role with no formal standards or certification, and a modicum of training in the actual practice of cross functional DevOps practices. They may already be experienced, highly skilled, competent and high performers in their own field of Software Development, Computing, IT Management, or Quality Assurance but they can lack the knowledge and understanding of the other cross functional disciplines they now find themselves working with daily. Understanding not only the technical, but also the business and human factors at play during the high pressure demands of modern software delivery processes, is essential in the modern discipline of DevOps.

Award Level

There are two phases to the award. Candidates are registered for the full Masters of Science in DevOps Level 9 degree (90 credits) however candidates may opt to exit the programme on successful completion of the first three semesters with 60 credits and receive a Level 9 Postgraduate Diploma in DevOps (60 credits). Please note exit awards are at the discretion of the college and no refund of fees will be due.
The award structure will place greater emphasis on continuous assessment, practical and project work rather than on formal examinations. In fact there are only 2 modules that carry an actual exam.
The aim is that participants will gain a deep understanding of the topics and content covered, and be able to demonstrate this acquired knowledge as proven competence in tests and exercises drawn from practical "real life" DevOps scenarios.

Programme Delivery

The programme will start with a 3 day workshop which will involve all participants being physically present. This is seen as important to facilitate networking, experience sharing and group learning.

It is expected that lectures will be delivered one evening per week in term time and every 3-4 weeks there may be a requirement to hold lectures twice in that week. There will also be a requirement to attend one on-campus day at the end of each semester.

Lectures will be streamed live from TU Dublin (Tallaght Campus) i and will be available for download and offline viewing.

Semester 1: Introduction to DevOps

Human and Organisational Issues	Software Development Methodologies
• Lean and Agile movements and methods • Assess and evaluate organisational design and culture to facilitate DevOps style development, deployment and support • Develop and manage global multi-disciplinary teams including an understanding of the cultural and practical issues which arise • Be able to form, lead and develop teams • Assess competence, accountability, responsibility, norms and operational management • Collaboration, negotiation and partnering • Managing the Future - Creating a readiness for organisational change, organisational development and change management	• Technical implications of DevOps – the philosophy, the history, the SDLC, Lean, Agile Manifesto, continuous feedback and learning • Change, Source, Defect Control Systems, Examination of major industry implementations (e.g. Atlassian, VSTS) • Code Promotion • Code Synchronization • System Debugging • Software QA • Automated Testing • Software Security Vulnerability Management • Software Telemetry and Monitoring • Feedback and Learning

Semester 2: DevOps Fundamentals

Business Technology Strategy	IT Infrastructure Fundamentals for DevOps
• The Business Case for Agility and DevOps • Lean/Agile management/methods/frameworks (SAFE) • Product road maps, pipelines, backlogs, valuing new features and technical debt • Business case development and risk assessment • Creation/management of multi-annual business plans • Financial Management of Product and Technology life-cycles • Project Management and Methodologies • The end of the monolithic project • Designing for agility and value • Challenges for DevOps • Regulated Software • Impact for Customers of DevOps approach	• Automation of Infrastructure • Task and Process automation languages • Advanced System Administration • Software Security • System Hardening • Policies and implementation • Virtualisation • Containerisation • IT Network and Infrastructure Protocols • IT Network Monitoring • Continuous Deployment • Cloud Computing Concepts • Infrastructure as Code

> *Understanding not only the technical, but also the business and human factors at play during the high pressure demands of modern software delivery processes, is essential in the modern discipline of DevOps.*

Semester 3: Advanced DevOps

Advanced IT Infrastructure for DevOps	DevOps in Practice
• Architectural Design to support DevOps • The DevOps supply-chain and PLM relationship • DevOps in the Public Cloud • Comparative Analysis of Cloud Offerings • Cloud Scalability and Elasticity3 • Load Balancing • Virtualisation Automation • Provisioning and Orchestration • Software Configuration Management • Software Provisioning Management • Security in the Public Cloud • Degrading systems gracefully • Chaos Monkey • Server-less Compute in the Cloud	• The DevOps paradigm/pipeline in practice requirements • Develop Continuous Integration/Test/Deployment Release management • Monitor and Learn • Feedback and Iteration • Detailed DevOps Case Study of the technical and human experiences of typical practitioners, e.g. - Google SRE (Site Reliability Engineering) - Intercom (Customer Messaging Platform)

</Code>

Semester 4: DevOps Research

Research Methods	Research Project
• Academic Writing • Qualitative and Quantitative research • Surveys • Statistics	• Applied piece of Research in DevOps area • Encompasses a Proof of Concept/Prototype • Supplements DevOps Theory knowledge *This is an opportunity for students to carry out a piece of work which is at the cutting edge of the field and explores in depth a feature or element of that field. It is perfectly feasible, and there are many examples of this, for students to carry out their research project on a piece of work of direct relevance to their company or organisation. The academic team in TU Dublin (Tallaght Campus) have deep industry experience and have supervised and developed MSc. projects which explore business values, infrastructure automation and DevOps projects with real industry relevance.*

- ## M. Sc. Applied IT Architecture (online)

In conjunction with Irish Computer Society and accredited by International Association of Software Architects. A Technology Ireland Skillnet funded programme. This programme is 80% online with two days per semester attendance required.

- ## M. Sc. Computing with DevOps (online)

This programme was designed in conjunction with leading ICT companies such as Microsoft, Fidelity, IBM, Ericsson who form the Technology Ireland Skillnet. This programme is 80% online with two days per semester attendance required.

Non-Standard Applicants

Note for interested applicants: Next intakes for these Skillnet programmes set for January 2019. Standard admissions requirements include a relevant bachelor's degree at honours level. It is recognised that there are experienced and skilled potential participants for the programme who may not fit the standard entry profile. A non-standard admission process is available here which can be based on prior experiential learning and/or qualifier modules. These qualifier modules can be taken from September 2019 for admission in January 2019. Contact bfeeney@it-tallaght.ie or mhendrick@it-tallaght.ie for more information.

Accreditation of Master of Science in Applied IT Architecture by IASA

Technology Ireland ICT Skillnet

The TUDublin (Tallaght Campus) M.Sc. in Applied IT Architecture is the first of its kind in the world to be developed based on the IASA Five Pillars.

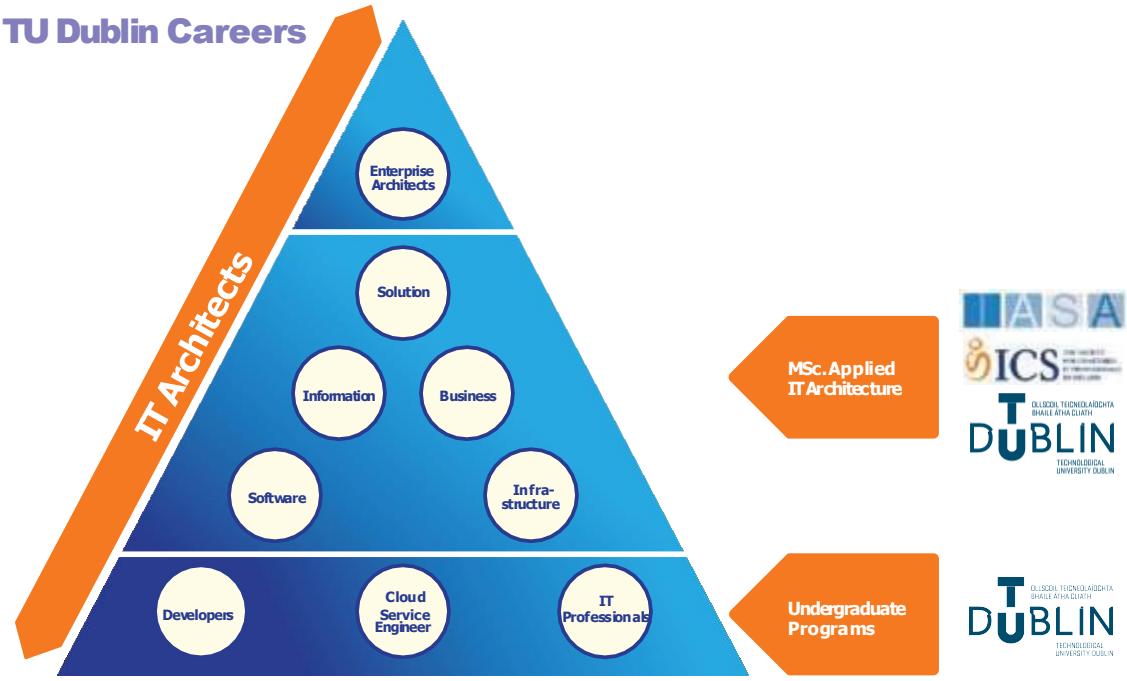
For the first time, candidates can gain a full Masters of Science degree in this specialist area through a mixed learning process with an emphasis on practical application in the workplace.

What is IT Architecture? (From IASA Global)

Architecture at IASA is the practice of business, organization or client gain through the application of technology strategy. It is the art and science of designing and delivering valuable technology strategy. At its core, the ITABoK describes how to create a professional person or group of professionals who can consistently find new applications of technology to generate positive outcomes for their client or employer. IT Architects:

- Retain depth in technical skill as well as business skill
- Able to successfully work with both business and technical staff
- Develop their own or others business cases based on technology driven innovation
- Retain the ability to deliver projects on those business cases
- Deliver business projects more successfully based on outcomes than others

TU Dublin Careers



PROJECTS

Interoperability and Standardization of Carbon-Aware Practices in Cross-Cloud Situations

Szabolcs Varga
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218709@mytudublin.ie

Introduction

In the past ten years, cloud computing has grown exponentially and Data centers use up over 415 TWh of electricity each year. This is about 1,5 percent of the world's electricity consumption. Carbon-aware computing aims at aligning digital workloads with low-carbon energy availability, but it has been hindered by fragmented implementations and inconsistent standards among cloud providers. The thesis examines interoperability challenges in cross-cloud carbon-aware computing and proposes a framework for portable and verifiable workload scheduling across Google Cloud, AWS, and Azure **using an API application created by the author**. Through a literature-review, abstraction-layer design and empirical prototype evaluation, the study delivers guidelines and an open-source toolkit that allows practical multi-cloud carbon-aware scheduling.

Research Question and Hypothesis

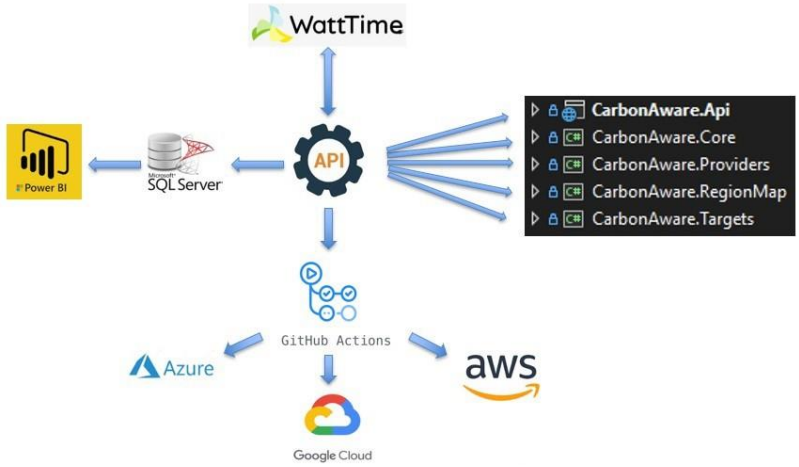
Research Question: How can carbon-aware workload scheduling be made interoperable and standardized across major cloud providers to promote scalable and sustainable computing?
Hypothesis: By creating a vendor-agnostic abstraction layer that standardizes carbon in-tensity data formats, workload policies, and scheduling APIs, organizations can enable cross-cloud carbon-aware workload orchestration that is both technically feasible and environmentally beneficial.

Carbon Signals: Why MOER?

Carbon-aware scheduling is all about making smarter decisions that consider the real im-pact of our actions on the environment. This approach utilizes the **Marginal Operating Emissions Rate (MOER)** from **WattTime**, which measures the extra carbon emissions that result from increasing electricity demand at a specific time and place so the system can bet-ter pinpoint times when it's more eco-friendly to run workloads. This means we can schedule tasks across different cloud environments ulti-mately helping to reduce our carbon footprint.

Solution Architecture

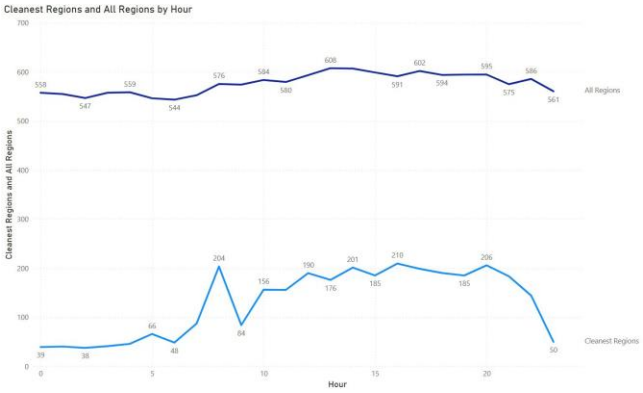
1. MOER signal acquisition from WattTime
2. API computation - advise
3. Github Actions call
4. IaC on the selected cloud provider
5. Logging/PowerBI - optional for troubleshooting/reports



Testing Results

Testing Method Summary: the system was tested over three full 24-hour cycles, covering all available cloud regions. Each cycle ran the same workloads using different scheduling modes to allow fair comparison. Results were then analysed to understand emission savings, timing behaviour, and regional selection patterns. **First figure** shows the CO2 emissions of cleanest regions the application selected versus an average baseline over 24 hours timespan and the **second figure** shows the cleanest regions identified over the experimental runs.

CO2 emissions (g/Kwh) Cleanest vs All Regions



Top 10 Cleanest Regions Identified

Conclusions and Future Work

This thesis demonstrates that carbon-aware workload scheduling is both viable and effective. The Carbon-Aware API tool demonstrated how the carbon exposures could be dramatically reduced (up to 90%) automatically by exploiting a temporal and geographic differential in marginal CO2 emissions. Such results corroborate recent literature but add real world tools and practices for sustainable DevOps. Future work should look at running the system over longer periods to better capture seasonal trends, balance carbon savings with cost and latency, integrate with production orchestration systems, and directly measure emissions from live workloads.

QR Code for Recording

Comparative Study: Infrastructure as Code Using Prompt Engineering Techniques.

Fergal Connolly
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218711@mytudublin.ie

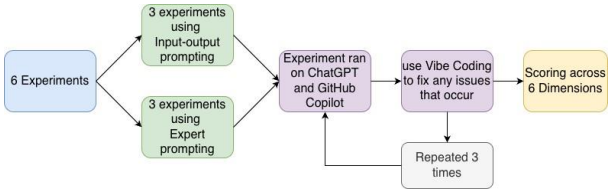
Introduction

Infrastructure as Code (IaC) has transformed the design, deployment, and management of cloud environments. Since 2024, the emergence of Large Language Models (LLMs) such as ChatGPT and GitHub Copilot has transformed software development practices. These platforms have made advanced generative AI technologies readily accessible. This study examines whether LLMs can generate IaC that is functionally correct, secure, and comparable to human-written Terraform, and whether prompt engineering can improve the quality. To investigate this, six experiments were carried out using ChatGPT and GitHub Copilot, with each introducing additional architectural complexity. The overall results were compared against a manually authored baseline, with evaluation focused on maintainability, compliance, and presence of hallucinations.

Research Question

RQ1: Can prompt-engineered IaC generated by an LLM match or outperform manually written Terraform configurations.
RQ2: Can non-experts effectively use LLMs to produce functional, high-quality IaC?

Method

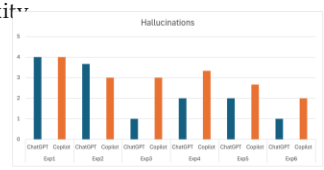


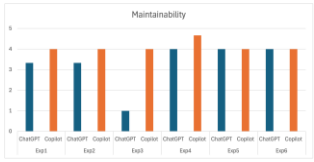
Experiments were conducted using an incremental infrastructure complexity model and executed three times per LLM to ensure result consistency and to mitigate the effects of output generation. IaC artifacts were produced using structured natural language prompts based on Input–Output Prompting and Expert Prompting techniques. A total of six prompts were designed to provision small, medium, and large Azure Kubernetes Service (AKS) clusters, with each introducing additional architectural elements as complexity increased. The analysis focuses on quality dimensions relevant to IaC: **Infrastructure Deployment**, **Hallucinations**, **Linting and Formatting**, **Monitoring Deployment**, **Maintainability**, and **Compliance**. This approach enabled the evaluation of how LLMs perform as the scenarios increased in complexity.

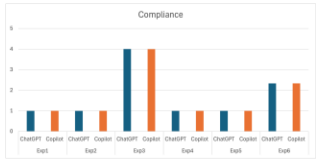
OpenAI ChatGPT and GitHub Copilot were both evaluated, and applying a technique called vibe coding, whereby the LLMs were prompted iteratively in response to validation and deployment errors. IaC quality was assessed using Terraform validation to verify syntax and structural correctness, while Terrascan was used to evaluate security posture and policy compliance. Following successful infrastructure provisioning, a lightweight Grafana monitoring stack was deployed to confirm operational readiness. A manually authored IaC implementation served as the baseline for comparison, allowing for the assessment of technical accuracy and consistency across workflows.

Results

Large Language Models can rapidly generate functional IaC for low-complexity deployments, making them effective tools for scaffolding and boilerplate generation. However, as architectural complexity increases, reliability, contextual consistency, and compliance degrade significantly, with recurring security and governance failures across all experiments. Sample results, evaluated on a five-point scale (where 5 denotes no issues and 1 denotes serious issues or failure in the generated code), demonstrate a decline in LLM effectiveness as infrastructure requirements increase in complexity.

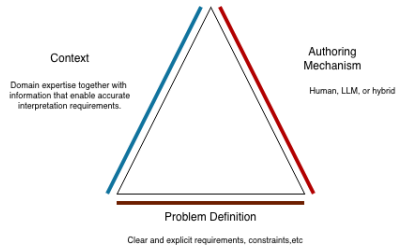






While AI-generated IaC accelerates development, and prompt engineering techniques enhance the structure and clarity of generated code, it also requires sufficient domain knowledge to craft prompts that effectively leverage LLM capabilities. LLMs can lower the barrier to entry, enabling non-experts to participate more readily in IaC development. However, expert domain oversight remains essential. Without it, the risk of insecure, non-compliant, or unstable infrastructure increases significantly, especially in more complex environments.

To synthesise these findings, this study proposes the IaC Dependency Triangle, a conceptual model that explains Infrastructure as Code quality as the interaction of three factors: problem definition, contextual knowledge, and the authoring mechanism. When requirements are clearly defined, context is stable and well understood, and the code creation method is reliable, IaC outcomes are significantly more consistent and secure. Conversely, weaknesses in any one dimension, such as ambiguous requirements, missing platform context, or unreliable AI generation, can lead to hallucinations, architectural drift, or security gaps. The model provides a practical framework for reasoning about manual, AI-assisted, and automated IaC workflows, while highlighting the continued need for expert oversight in complex environments.



Conclusions and Future Work

This study demonstrates that LLMs can generate deployable Terraform code for simple architectures when requirements are clearly defined and prompts are well-structured, but hallucination and issues increase as architectural complexity increases. Compared to AI-generated code, manually authored Terraform remains more stable, secure, and predictable, although it is more time-intensive. The proposed IaC Dependency Triangle demonstrates that IaC quality depends on problem definition, context, and the authoring mechanism. Overall, LLMs are valuable assistive tools, but expert oversight remains essential for complex infrastructure.

Future research should examine how developers with different levels of experience interact with LLM-generated IaC and evaluate performance. The use of AI-generated code also raises ethical concerns, particularly in academic and professional settings, which require further study. In addition, validation of the IaC Dependency Triangle is needed.

QR Code for Recording

Evaluating Cloud Speech-to-Text Services

Luke Connolly
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218713@myTUDublin.ie

Cloud speech-to-text (STT) services promise accuracy, scalability and low-cost transcriptions, yet vendor accuracy is not widely reported and many available benchmarks use clean, lab-like audio for these accuracy figures. A key question is how these systems perform under real-world conditions, where speech is often degraded by background noise and interference. This thesis presents a comparative evaluation of Amazon Transcribe, Microsoft Azure Speech and Google Cloud Speech-to-Text, focusing on accuracy, latency and cost to provide a trade-off analysis for organisations. Understanding these trade-offs allows organisations seeking to embed STT into production workflows to better understand each vendor's strengths.

Research Question

Which cloud speech-to-text service offers the best trade-off between accuracy, latency and cost when transcribing Irish English speech across varying acoustic conditions?

Methodology

- **Create a reproducible, DevOps-ready evaluation framework:** Implement an automated benchmarking pipeline using vendors' official SDKs, built as modular Ruby scripts and designed for CI/CD integration.
- **Experiment design:** Start with clean Irish English speech audio and apply pink noise to generate controlled acoustic conditions (Clean, 20 dB, 10 dB, 0 dB SNR).
- **Evaluate leading vendors:** AWS Transcribe, Microsoft Azure Speech and Google Cloud Speech-to-Text.
- **Benchmark metrics:**
  - **Accuracy:** Word Error Rate (WER), error-type breakdown (substitutions/deletions/insertions), and misrecognition analysis.
  - **Latency:** Time from job submission to completed transcription.
  - **Cost:** Cost per hour and scaled production workload projections.

Key Findings

Accuracy is consistent across all conditions: **AWS** (1st), **Azure** (2nd), **GCP** (3rd).

Accuracy (Word Error Rate)

- **WER (Clean):** AWS 4.34%, Azure 10.27%, GCP 17.58%.
- **WER (0 dB):** AWS 8.22%, Azure 25.11%, GCP 32.88%.
- **Overall mean WER (all conditions):** AWS 5.71%, Azure 15.13%, GCP 22.49%.

Error fingerprints (What causes the error increases)

- Degradation in quality of transcriptions is **substitution-led** for all vendors.
- **Azure:** Larger **deletion** rise at heavy noise.
- **GCP:** More **insertions** at mid-noise (10 dB), suggesting occasional short-word hallucinations.

Latency

- Total latency: AWS 17.35 s, GCP 41.39 s, Azure 51.24 s → AWS is ~2.4–3× faster.

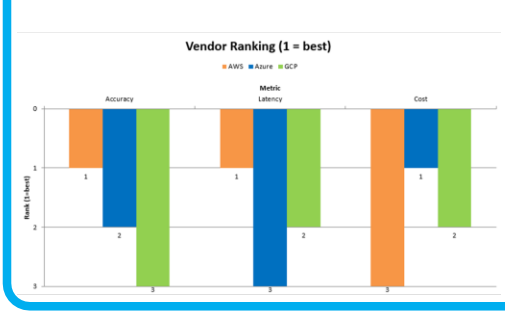
Cost

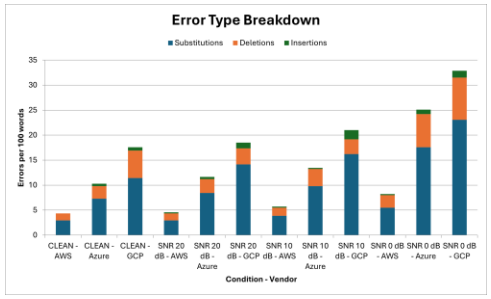
- AWS €1.24/hr, GCP €0.83/hr, Azure €0.16/hr → Azure is the cheapest cost option.

Trade-off Analysis

- Choose **AWS** when **quality** or **turnaround** matters most.
- Choose **Azure** when **budget at scale** dominates and higher WER/slower turnaround is acceptable.
- Choose **GCP** only when accuracy needs are looser or when there is data residency/on-premise constraints.

Topic Overview





Conclusions and Future Work

Results show that **AWS** delivers the strongest accuracy and the fastest turnaround, while **Azure** offers the lowest cost but with higher error rates and longer latency. **Google** trails on all benchmarks. The key takeaway is that organisations should evaluate vendors on their own audio and conditions, then select the service whose accuracy–latency–cost best matches the target use case.

Future Work: Will extend the evaluation to real-time streaming transcription, broader speaker styles to also include multi-speaker overlap and to use the pipeline for continuous regression testing in CI/CD to track performance changes across models, configuration and region updates. Additional experiments will also assess the value of custom vocabularies and review human editing effort required to reach usable transcripts for each vendor.

QR Code for Recording

A Comparative Analysis of AWS EKS and Azure AKS: Performance, Cost, and Environmental Efficiency for Containerised Microservices

Patrick Culligan
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218712@myTUDublin.ie

Introduction

Kubernetes has become the dominant platform for orchestrating containerised microservices, enabling scalable, resilient, and portable cloud-native applications. To reduce the operational complexity associated with running Kubernetes in production, cloud providers offer managed Kubernetes services that abstract control-plane management, automate maintenance tasks, and integrate closely with cloud-native infrastructure.

Among these services, AWS Service (EKS) and Azure Kubernetes Service (AKS) are two of the most widely adopted platforms. Although both expose the same upstream Kubernetes API, they differ in underlying infrastructure design, resource management strategies, pricing models, and optimisation mechanisms. These differences can influence application performance, operational cost, and environmental efficiency.

As organisations increasingly deploy microservices at scale and face growing pressure to optimise cost and reduce carbon emissions, understanding how managed Kubernetes platforms behave under realistic workloads is critical. This research provides an empirical comparison of EKS and AKS, examining their performance, cost efficiency, and environmental impact to support evidence-based platform selection for cloud-native applications.

Objective

The objective of this research is to compare AWS EKS and Azure AKS across performance, cost, and environmental impact using controlled microservice workloads, in order to support evidence-based cloud platform selection..

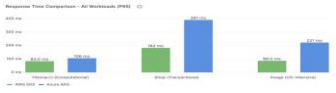
Research Question

Which managed Kubernetes platform—Amazon EKS or Azure AKS—provides the more effective environment for hosting containerised microservices when evaluated across performance, cost efficiency, and environmental impact?

Methodology

- Three .NET microservice workloads: compute, transactional, I/O
- Identical single-node EKS and AKS clusters
- Standardised deployment, configuration, and load conditions
- k6 load testing; Prometheus/Grafana for resource metrics
- Kepler for power + CO2 estimation
- Cloud pricing used for cost-per-request analysis
- Repeated trials for statistical validity

Results



Latency (P95)

Pod CPU

Node CPU

Power & CO_2

Throughput

Pod Memory

Node Memory

AVERAGE COST RESULTS

Platform	Compute (€ / hour)	Control Plane (€ / hour)	Total (€ / hour)	Total (€ / month)	Total (€ / year)
AWS EKS	€ 0.20	€ 0.10	€ 0.29	€212.72 / month	€2,552.66 / year
Azure AKS	€ 0.22	€ 0.00	€ 0.22	€158.78 / month	€1,904.70 / year

Cost

Conclusions and Future Work

AWS EKS demonstrates superior performance, scalability, and energy efficiency across the evaluated microservice workloads, making it well suited for production environments that require high throughput, low latency, and stable execution. Azure AKS remains a viable option for lightweight or cost-sensitive deployments due to its lower entry-level infrastructure costs, but its efficiency and stability diminish as workload intensity increases. These findings highlight that managed Kubernetes platform selection must be workload-aware, requiring performance, sustainability, and cost to be considered together rather than in isolation. Overall, the results answer the research question by showing that AWS EKS provides the more effective managed Kubernetes platform for production microservices demanding performance, reliability, and sustainable resource utilisation.

Future work will extend this study by expanding the platform comparison beyond AWS EKS and Azure AKS to include additional managed Kubernetes services and by evaluating identical workloads across multiple geographic regions. The environmental analysis can be enhanced by applying energy instrumentation consistently across all workload types and integrating region-specific carbon intensity data to improve emissions accuracy. Further research should also investigate more com-plex microservice architectures and analyse the transactional instability observed on Azure AKS through targeted resilience and failure-scenario testing, enabling a deeper understanding of platform behaviour under realistic production conditions.

QR Code for Recording

A Retrieval-Augmented Generation Approach to Conversational Analytics over Structured Data

Brian de Búrca
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218725@myTUDublin.ie

Introduction

Integrating Large Language Models (LLMs) with Retrieval-Augmented Generation (RAG) using a relational database can enable a chatbot to answer queries grounded in structured organisational data. This study examines three key aspects: (1) the accuracy and faithfulness of generated responses when compared with baseline results derived directly from structured data, (2) system responsiveness and usability for non-technical administrative and managerial users, and (3) the practical feasibility of adopting and maintaining such an approach within resource-constrained organisations.

Motivation

The motivation for this research is to evaluate whether organisations with limited IT budgets and resources can successfully adopt AI through a low-cost solution. The proposed approach aims to be easy to implement, use, and main-tain, without requiring significant financial or technical investment. As AI becomes an increas-ingly integral part of everyday business opera-tions, this study seeks to provide smaller organ-isations with insight into the potential value of AI by demonstrating a low-cost, low-code solu-tion that can be effectively managed in-house.

Research Questions

1. Can a RAG-based chatbot accurately interpret user input and generate logically consistent responses based on the source data?

2. Does the chatbot produce consistent responses to semantically similar queries while correctly maintaining conversational context?

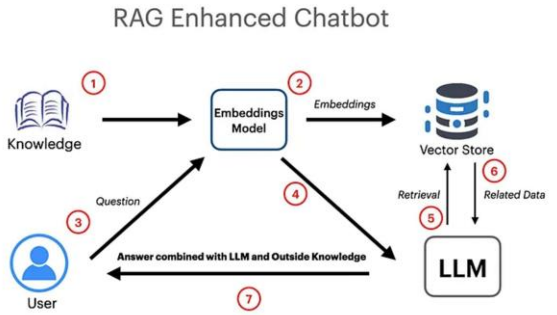
3. Can the solution be implemented and maintained at a low cost while remaining scalable?

How a RAG Chatbot Works

The main objective of a RAG chatbot is to interpret a user's query and retrieve relevant information to generate an accurate and context-aware response.

1. Knowledge and Embeddings

Knowledge is extracted from a **relational database** and transformed into numerical representations using an embedding model, such as *text-embedding-3-small* from OpenAI or *NV-Embed-v1* by NVIDIA. These embeddings capture the semantic mean-ing of the data for efficient retrieval.



2. Vector Store

The generated embeddings are stored in a vector database, such as AstraDB. Each data entry is assigned a vector representation, enabling similarity-based searches to efficiently retrieve the most relevant information for a given query.

3. Large Language Model

The Large Language Model (LLM) combines the user's query with the relevant results retrieved from the vector database. Using this contextual information, the LLM generates a coherent and natural-language response tailored to the user's question.

Building out the chatbot

Step 1. Create a vector database using astra.datastax.com.

Step 2. Populate AstraDB with data extracted from three sources representing different business contexts:

- Election data
- Student grades
- Electronic store sales (small openai embedding)
- Electronic store sales (Large openai embedding)

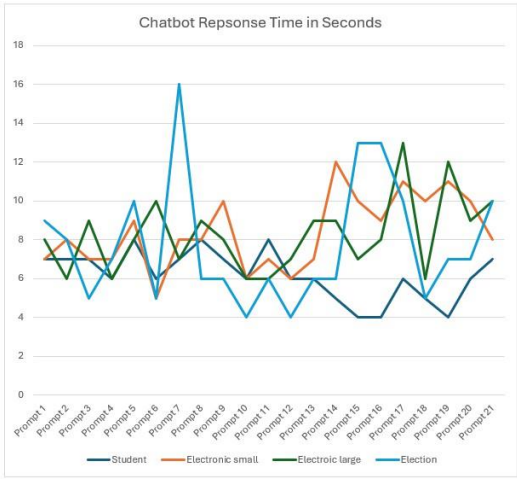
Step 3. Create a VM to host the Langflow application from a container.

Step 4. Build one flow per data source for testing.

Step 5. Evaluate each flow based on:

- Query response time
- Response quality compared to the original data source
- Anomalies in result sets

The image is the response times for each prompt given the different datasets



Conclusions and Future Work

The results indicate that the chatbot can generate contextually relevant and interpretable responses, demonstrating the potential of low-code, LLM-based systems for organisational access to structured data. However, performance is less reliable for aggregation-heavy and numerically precise queries. These limitations are primarily linked to embedding-based retrieval, chunking strategies, and top-*k* selection. For the student dataset, response accuracy improved when grade-related queries were routed to a direct database/API call rather than relying solely on retrieval from the vector store. This approach improves reliability for count-based and numeric queries while maintaining conversational quality for general queries.

Future work will explore improved embedding and chunking strategies to increase retrieval precision and response faithfulness. Further evaluation will also compare alternative low-code platforms against the Langflow implementation in terms of accuracy, usability, and cost.

QR Code for Recording

Autonomous IT Incident Resolution through AIOps with Local LLMs: A Practical Framework and PoC

Konrad Soares
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00218708@mytudublin.ie

Introduction

Problem. Modern monitoring detects incidents but rarely closes the loop. Engineers face alert fatigue, inconsistent runbooks, and long MTTR, especially in constrained or air-gapped environments.
Aim. Design and evaluate a *policy-gated, local-LLM* framework that goes from alert → triage → safe action → verification with full auditability and zero data egress.
Approach. Prometheus/Alertmanager trigger a Python orchestrator with a local LLM (Ollama). Policies allowlist actions and enforce budgets/cooldowns. Outcomes are recorded in `/state/case-*.json` and surfaced via Telegram.

Framework / Policy Model

1. Closed-loop design

Alert → Case → LLM triage → Policy gate → Actions → Resolution (alert clears) or Escalation.

2. Policy model

Allowlist per hint (e.g., frontend_down, disk_full); restart budgets; cooldowns; max attempts; unsafe-action blocks; optional silencing to avoid storms.

3. Execution flow

Triage proposes a plan (e.g., restart; prune → df → restart). Policy validates scope/limits; dispatcher executes; verification via cleared alert; otherwise escalate and draft runbook.

PoC Environment & Tooling

1. Stack

Prometheus + Alertmanager (webhook) · Python orchestrator · Ollama (local LLM) · Frontend + sidecars · Telegram · `/state/case-*.json`.

2. Config & policies

Alert labels → policy hint; YAML allowlist; budgets/cooldowns/max-attempts; env ports/thresholds; silencing rules.

3. Reproducibility & security

Deterministic scenarios (Appendix E); zero egress; least privilege; timestamps for MTTA/MTTR and audit.

Observability, Scenarios & Metrics

1. Observability

System-of-record: `/state/case-*.json` (actions, notes, lifecycle).
Operator timeline via Telegram.
Alertmanager state used as verification source.

3. Metrics

Detection latency (startsAt-T0), MTTA, MTTR, Autonomy %, guardrail compliance (attempts, budgets, cooldowns).

2. Scenarios (Appendix E)

S1: ENOSPC. Prune → df snapshot → restart; resolve on alert clear.
S2: Missing container. 3 attempts → escalate; runbook draft.
S3: FrontendDown. Restart with backoff; dedupe/suppression.

Key Results (KPIs)

- **MTTR:** Reduced in covered paths (ENOSPC, FrontendDown).
- **Autonomy %:** High where runbooks/policies exist; escalate on missing.
- **Noise:** Fewer repeats via silencing/dedupe.
- **Audit:** `/state/case-*.json` (actions, notes, lifecycle).

Safety & Governance

- Allowlist per hint (frontend_down, disk_full).
- Budgets, cooldowns, max attempts; restart caps.
- Escalate on denial or failed verification.
- Zero egress (local LLM); least privilege.

How to Reproduce (PoC)

$ docker stop frontend1

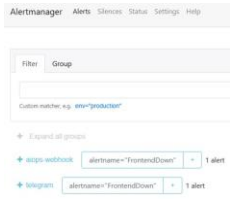


Fig. 1 Alert firing

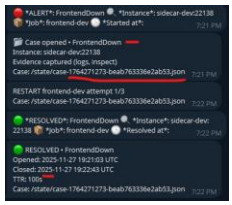


Fig. 2 Telegram message / Case / MTTR

Topic Overview

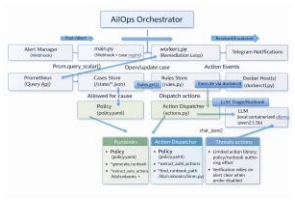


Fig. 3. System architecture

Conclusions

Conclusions.

- Closed-loop, policy-gated automation reduced MTTR in covered cases.
- Zero data egress with full audit trail (`case-*.json`) suits edge/air-gapped environments.
- Guardrails (allowlist, budgets, cooldowns) bounded risk; unresolved paths escalated with low noise.

QR Code for Recording

An Empirical Analysis of Query Router Scalability in Sharded MongoDB Clusters

Adeniyi Babatope
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218722@mytudublin.ie

1. Introduction

As distributed databases become the backbone of modern large-scale applications, the ability to scale horizontally is paramount. While the scalability of the data layer (shards) is well-documented, the performance implications of the routing layer, specifically the 'mongos' process remain under-explored. This thesis presents a rigorous empirical analysis of query router scalability in sharded MongoDB clusters, examining the effects of scaling 'mongos' nodes under high-concurrency workloads to identify precise saturation points.

2. Problem Statement

- **The Problem:** Industry practices often rely on "rule of thumb" heuristics for provisioning routers, leading to either performance bottlenecks (under-provisioning) or resource wastage (over-provisioning).
- **The Gap:** Lack of empirical data quantifying the saturation point of a single 'mongos' process under high concurrency.
- **Research Question:** *"How does horizontally scaling the number of 'mongos' routers affect throughput and latency across varying shard counts (1 to 9)?"*

3. Methodology

A strictly controlled **Factorial Design** experiment was conducted using Infrastructure as Code (IaC) to ensure reproducibility.

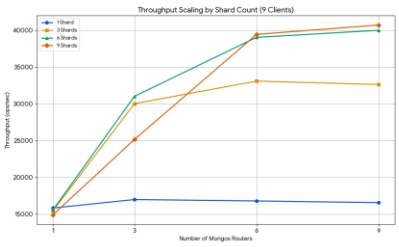
Experimental Setup

- **Infrastructure:** AWS c5.xlarge (Compute Optimized) for all nodes.
- **Orchestration:** Terraform Ansible.
- **Workload:** YCSB Workload A (50% Read / 50% Update).
- **Dataset:** 1,000 Records (Fits in RAM) to isolate CPU/Network performance from Disk I/O.
- **Sharding:** **Hashed Sharding** on _id to guarantee uniform distribution and prevent "Hot Shards."

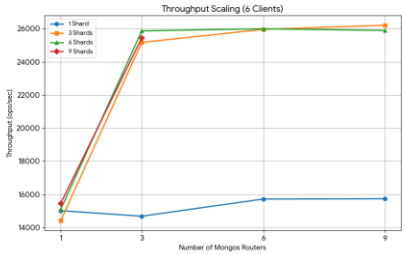
Matrix of Variables:

Variable	Values
Shards	1, 3, 6, 9
Mongos Routers	1, 3, 6, 9
Concurrent Clients	1, 3, 6, 9

4. Key Findings: The "Rule of 6"



Peak Load (9 Clients): Throughput plateaus at 6 Routers.



1-Shard Saturation: Adding routers yields zero gain.

Key Findings:

- **The "Rule of 6":** In a 9-Shard cluster under peak load (40k ops/sec), scaling from 1 to 6 routers yielded linear gains. However, scaling from **6 to 9 routers provided <2% improvement**, identifying the saturation point.
- **1-Shard Ceiling:** For a single-shard cluster, throughput flatlined at ≈16k ops/sec regardless of router count. This confirmed **Logical Contention** within the WiredTiger storage engine (locking overhead) was the bottleneck, not the routers.
- **Client-Bound Baseline:** Single-client tests flatlined at 4,500 ops/sec, proving that low-concurrency workloads are latency-bound (Amdahl's Law) and cannot benefit from horizontal scaling.

5. Impact of Consistency

A secondary analysis compared the Load Phase ('w=majority') vs. Run Phase ('w=1').

Mode	Throughput	Latency
w=1	≈ 4,500 ops/sec	2.3 ms
w=majority	≈ 1,350 ops/sec	6.0 ms

Insight: Strong consistency introduces a **3.3× latency penalty**, shifting the bottleneck from the router CPU to network replication lag.

6. Conclusions & Future Work

Conclusions

- **Diminishing Returns:** Over-provisioning routers (beyond a 1:1.5 ratio) wastes resources without improving throughput.
- **Bottleneck Shift:** The bottleneck dynamically shifts from *Client* (low load) → *Router* (medium load) → *Network/Locking* (peak load).
- **Efficiency:** The mongos process is highly efficient (<5% CPU usage), suggesting bottlenecks are due to network coordination, not compute.

Future Work

- **Sidecar Deployment:** Evaluating Client-Side mongos (e.g., in K8s pods).
- **Disk-Bound Tests:** Exceeding RAM to test "Thundering Herd" scenarios.
- **Client-Side vs. Server-Side Routing:** Investigating the performance impact and architectural trade-offs between client-routed and server-routed database queries.

Scan for Video

Open-Source Tooling in Cloud Environments for Lawful Interception Tools

Borja Marazuela
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218724@myTUDublin.ie

Introduction

This research addresses the need for cost-effective and transparent Lawful Interception solutions for network operators shifting to cloud environments. By establishing quantitative benchmarks for OpenLI in Google Cloud Platform (GCP), the study compares three deployment models: a single virtual machine (VM), a distributed architecture with dedicated VMs per service, and a containerized environment managed via Kubernetes. Performance was measured using synthetic VoIP traffic loads to evaluate latency, throughput, and packet loss.

Our results indicate that OpenLI's performance scales linearly with increased CPU cores, though it is currently bottlenecked by its single-instance mediation function and the capacity of Law Enforcement Agency (LEA) systems. Ultimately, our testing found the distributed VM-per-service model to be the most efficient and performant solution. It offers superior flexibility for resource optimization without the added complexity and architectural friction introduced by container orchestration tools.

Lawful interception

Lawful Interception (LI) is formally defined by the ITU as the "lawfully authorized interception and monitoring of telecommunications pursuant to an order of a government body" to obtain forensic evidence against wrongdoers. It rep-resents the technical and legal intersection of national security needs—such as investigating organized crime and terrorism—and the societal right to privacy. Modern LI has evolved from physical wiretapping of copper lines to the complex, automated capture of data in packet-switched networks, governed by frameworks like CALEA in the US and ETSI standards internationally.

State of the art

Current research in the LI field focuses on modernizing legacy systems through several key trends: Lawful Interception as a Service (LIaaS), which leverages cloud-native technologies for media analysis and intelligent extraction; Self-Sovereign Identity (SSI) to automate and secure interactions between agencies; and specialized mechanisms for 5G networks to address end-to-end encryption challenges. OpenLI itself represents a major step forward as an ETSI-compliant open-source alternative that lowers entry barriers for network operators by reducing reliance on expensive, proprietary hardware

Empirical testing

1. OpenLI Component architecture:

OpenLI utilizes a microservices-based architecture consisting of three primary components:

- Provisioner: Acts as the administrative interface, receiving warrants manually through a web portal or automatically via an API and translating them for the network.
- Collector: Mirrors network traffic and filters for packets belonging to intercepted targets based on the list provided by the provisioner.
- Mediator: Encompasses mediation and delivery, adding ETSI-compliant headers and buffering data in a RabbitMQ queue for delivery to the Law Enforcement Monitoring Facility (LEMF).

2. VM environment:

Several VMs were tested with various configurations. For Scenario 1 (single VM), performance scaled near-linearly with CPU cores, with 2 vCPUs being the minimum required to prevent packet loss. Scenario 2 (VM-per-service) demonstrated that the Collector is the primary CPU consumer during processing, while the Mediator uses more memory to buffer retransmissions.

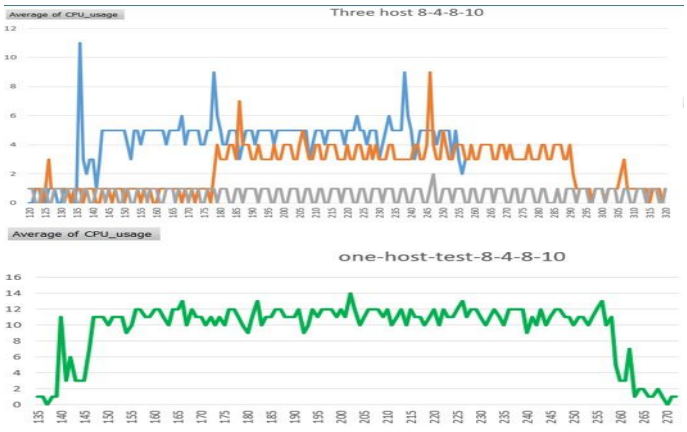
Stress testing revealed the system's limits; at this peak, the system began dropping packets because the collector could no longer handle the incoming traffic volume. There was no significant difference in total resource usage between the single-host and three-host VM setups.

3. Kubernetes environment:

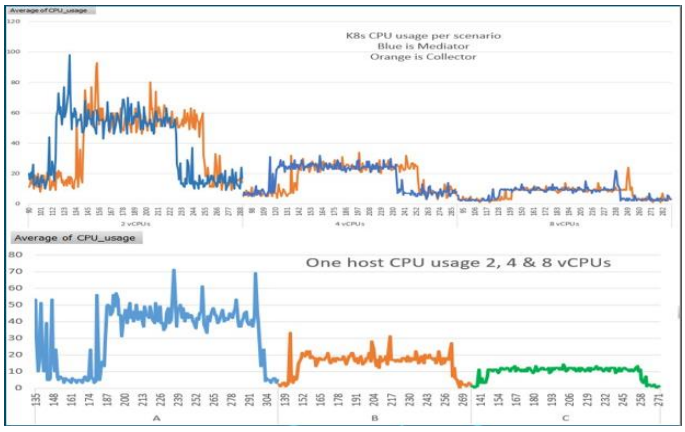
Scenario 3 utilized Google Kubernetes Engine (GKE) to deploy the three OpenLI services in individual pods, with scaling tested across different host configurations. While GKE provided high availability and self-healing, the deployment process was more complex due to the need to reconfigure local RabbitMQ instances upon every restart. In the 2 vCPU scenario, the system was at capacity suggesting higher overhead than the VM scenarios.

GKE results mirrored the VM performance in terms of linear scaling, but the environment introduced resource under-utilization due to the required parity between cores and RAM.

Graphic Results



(a) Host per service CPU usage comparison



(b) Kubernetes CPU usage comparison

Conclusions and Future Work

Our testing shows that separated VMs are currently the most practical and performant deployment method for OpenLI. This model allows for granular resource allocation. Future research lines are enhancing OpenLI by adding extra features and replicating this testing on other cloud providers.

QR Code for Recording

Observability of Continuous Delivery Pipelines with OpenTelemetry: A Case Study with Spinnaker

Noel McKeown
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218705@mytudublin.ie

Introduction

Continuous Delivery (CD) pipelines are the backbone of modern software release processes, yet many organisations lack meaningful visibility into how these pipelines perform. This absence of pipeline level observability creates a blind spot that limits operational awareness, impedes root cause analysis and restricts the ability to measure delivery performance objectively. This research investigates how OpenTelemetry (OTel), a vendor neutral observability technology can be extended beyond traditional application monitoring to instrument continuous delivery pipeline lifecycle events. Using the Spinnaker CD platform as a case study, the research demonstrates how native pipeline execution events can be captured, transformed into structured telemetry and exported as Prometheus compatible metrics. These metrics are further aligned with DORA performance indicators, enabling empirical measurement of deployment speed, reliability and recovery.

Observability Gap in CD Pipelines

Continuous delivery platforms remain under instrumented, leaving pipeline health, perfor-mance and failure behaviour poorly measured. This research identifies a critical observability gap where pipeline health, execution duration, failure patterns and recovery behaviour are not consistently measurable. Without pipeline met-rics, teams cannot proactively identify bottle-necks, assess release stability or support service level objectives. Addressing this gap is essential for data driven DevOps decision making.

OTel Integration Architecture

A prototype architecture was implemented to integrate Spinnaker with OpenTelemetry by capturing pipeline lifecycle events via a lightweight webhook receiver and processing them through an OpenTelemetry Collector. Events are nor-malised into consistent metrics for ingestion by Prometheus and visualisation in Grafana us-ing only open source components. The results demonstrate OpenTelemetry's viability as a uni-fied observability layer for continuous delivery

Mapping Pipeline Telemetry to DORA Metrics

A core contribution of this research is the mapping of raw pipeline execution events to DORA metrics. Deployment Frequency, Lead Time for Change, Change Failure Rate, and Mean Time to Restore. By extracting timestamps, execution states, trigger metadata and failure signals from pipeline telemetry, these metrics can be computed automatically in near real-time. Grafana dashboards and alerts provide actionable insights into delivery performance, enabling teams to correlate operational behaviour with business outcomes. This approach operationalises the pipeline metrics and provides a DORA scorecard for CD performance.

Deployment Frequency & Lead Time for Change

Pipeline execution events provide precise timestamps for when a deployment starts and successfully completes. These signals enable Deployment Frequency to be calculated directly from successful pipeline runs over time, while Lead Time for Change is derived from execution start to completion duration.

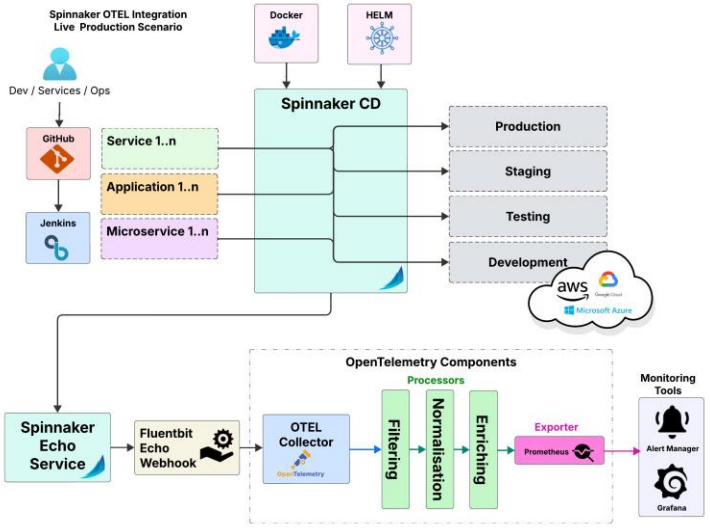
Change Failure Rate & Reliability Signals

Pipeline failure and cancellation events are transformed into structured metrics that quantify Change Failure Rate. Failed executions, rollbacks and aborted stages provide a direct signal of deployment instability and release risk.

Mean Time to Restore & Operational Recovery

Recovery related pipeline events enable calculation of Mean Time to Restore (MTTR) by measuring the interval between a failure event and the next successful execution. This captures how quickly teams can recover service using pipeline driven remediation or rollback workflows.

Architecture and DORA Dashboard



Conclusions and Future Work

Extending OpenTelemetry into CD pipelines is feasible and valuable, providing unified observability, actionable pipeline insights and objective DevOps performance measurement through an open source repeatable architecture.

Future work should validate the approach at enterprise scale, optimise telemetry performance, apply security standards and extend pipeline observability to other CD platforms and OpenTelemetry conventions.

QR Code for Recording

Ground truth Circuit-breaker: Leveraging Surrogate Models to Achieve Graceful Degradation in Distributed Systems

Benjamin Murray
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218726@myTUDublin.ie

Modern distributed systems increasingly rely on resilience mechanisms to cope with partial failures, overload, and unpredictable runtime conditions. Traditional fault-tolerance patterns, such as retries and circuit breakers, typically fail fast or degrade functionality when backend services become unavailable. This thesis explores a novel resilience approach, the Ground Truth Circuit Breaker, in which lightweight machine learning models are positioned as middleware in front of microservices to generate approximate responses when downstream services are unavailable or under extreme load. The research investigates the feasibility, performance characteristics, and architectural implications of replacing live service calls with ML-generated responses at runtime. The work evaluates this pattern as an extension to existing resilience strategies, with particular focus on system availability, latency, and applicability in high-load scenarios.

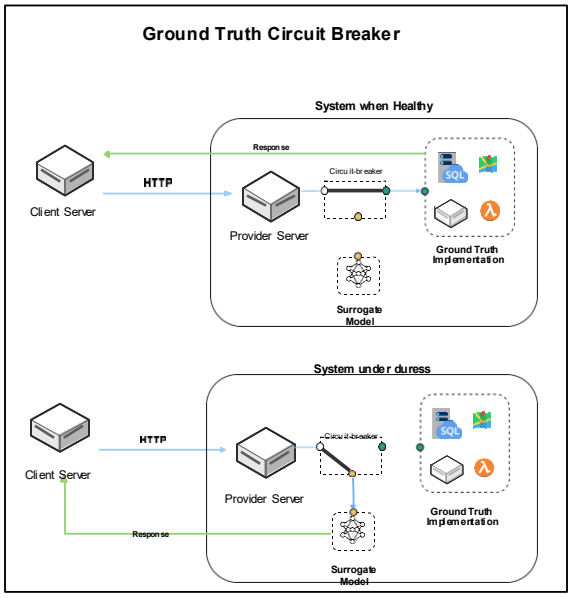
TOPSIS Analysis

RQ1: Can the *Groundtruth Circuit Breaker*, which uses a surrogate model to approximate downstream service behaviour, improve system resilience under partial or total service degradation?

This primary question is supported by the following subsidiary questions:

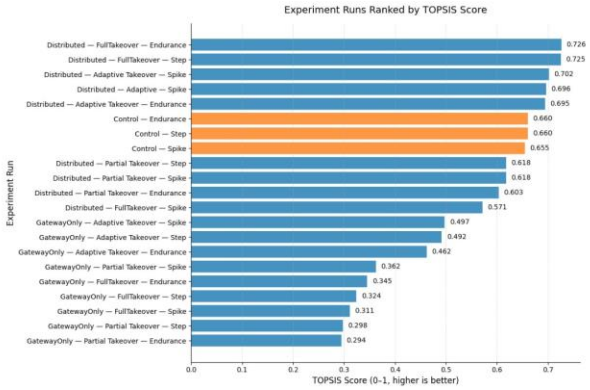
- **RQ1.1:** How accurately can a surrogate model approximate the outputs of a real service under normal operating conditions?
- **RQ1.2:** How does system behaviour vary under different traffic patterns, surrogate placements, and takeover policies?
- **RQ1.3:** How does surrogate-based fallback compare to traditional resilience mechanisms in terms of latency, error rates, and stability?
- **RQ1.4:** What limitations, risks, and trade-offs arise when using surrogate models for degraded response generation?

A New Resilience Pattern



When the system is healthy, surrogate models remain dormant while continuously learning from live production traffic. Under duress, the circuit trips and traffic is fully or partially routed to these models until normal operating conditions are restored. The pattern introduces a controlled trade-off, sacrificing some response fidelity in exchange for lower latency and higher throughput.

Results



Analysis

A weighted TOPSIS analysis showed that multiple configurations of the Groundtruth Circuit Breaker pattern achieved a favourable trade-off between performance and accuracy. The decision model prioritised output fidelity (×3) over throughput (×1.2), p95 latency (×1.4), and CPU utilisation (×1.01), reflecting the importance of response correctness under degradation. Under this weighting, several surrogate-based configurations not only reduced end-to-end latency but also outperformed the control baselines once their strict 100% ground-truth fidelity requirement was considered, demonstrating measurable resilience benefits with bounded approximation error.

Conclusions and Future Work

This thesis demonstrated that the Groundtruth Circuit Breaker, which uses surrogate models to generate approximate responses during service degradation, can significantly reduce latency and increase throughput in distributed microservice systems. The results show that effectiveness depends on architectural and operational choices, with service-local surrogate placement and adaptive takeover strategies providing the best balance between performance and fidelity, achieving accuracy levels of approximately 85–90%. While surrogate outputs do not fully match ground-truth responses, the resulting trade-off is acceptable in latency-sensitive systems that can tolerate reduced response fidelity. Future work includes extending the pattern to tolerate network-level failures and exploring generative model types to support complex, high-dimensional response formats.

QR Code for Recording

Green AI: Evaluating the Environmental Cost of Training Machine Learning Models

Sian O'Briain
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218714@myTUDublin.ie

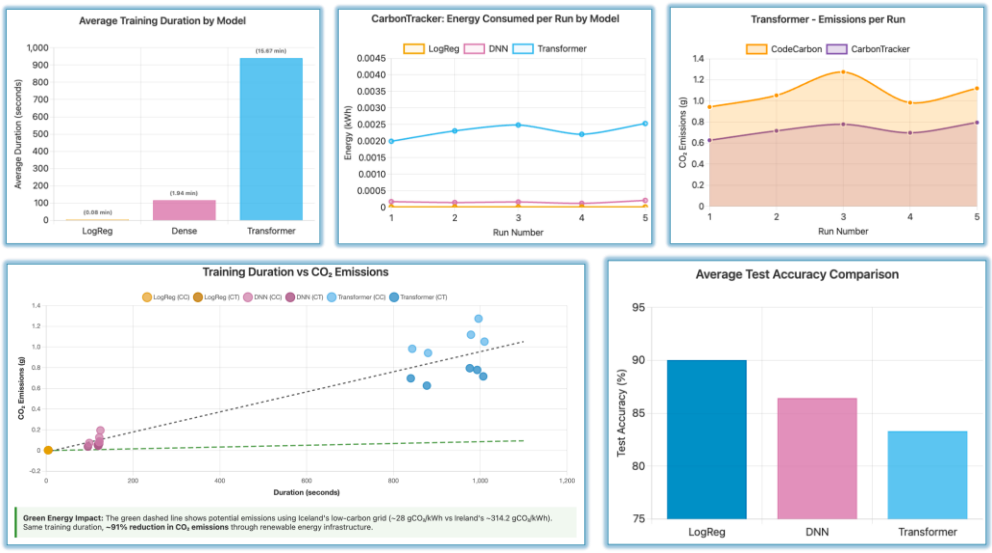
Introduction

As the use of AI continues to rise so does its corresponding carbon footprint. While much of the current research focuses on monitoring machine learning workloads running in large-scale cloud environments, this study aims to investigate the relationship between model complexity and associated carbon emissions during training in a local controlled environment. Three model types of varying complexity - Logistic Regression, a Dense Neural Network and a Tiny Transformer are trained using the same IMDb film review dataset and on the same binary classification task to determine if a film review has a positive or negative sentiment. Each model's performance is then evaluated in order to further explore the trade-off between predicted performance and associated carbon footprint. Energy consumption and carbon emissions are estimated using two carbon tracking tools, CodeCarbon and CarbonTracker which are also compared as part of the analysis.

Results

Energy & Emissions

The Transformer model dominated total energy consumption & accounted for >90% overall usage, emitting ≈300× more CO_2e than Logistic Regression, training duration being the primary contributing factor. Discrepancies were observed between carbon tracking tools due to differences in estimation methodologies. Variations in carbon intensity had a significant impact on the resulting emissions.



Performance Vs Energy Cost

Logistic Regression achieved the highest predictive performance with greater than 90% accuracy, while also being ≈300× more carbon-efficient than the Transformer model. For this task, the Transformer performed worse despite its substantially higher environmental footprint. Early stopping and patience avoided unnecessary emissions from continued training and prevented any performance degradation.

Conclusions and Future Work

Model choice matters
From a Green AI perspective, bigger is not always better, as larger models incurred significantly higher energy consumption and emissions in this study. These findings highlight that model selection has a direct and substantial impact on environmental cost.

CodeCarbon as the most suitable tracking tool
CodeCarbon was identified as the most suitable tracking tool due to its greater configurability and its consideration of system components beyond CPU and GPU usage. In addition, its structured output formats were well suited for analysis, and its active maintenance and community support contribute to its reliability.

Carbon intensity and geography strongly influence emissions
Identical workloads can result in vastly different emissions depending on the underlying energy mix and carbon intensity of the execution location. As training workloads are geographically flexible, this makes emissions optimisation both practical and feasible.

No single solution is sufficient
Model selection and workload location alone are not enough to address AI's environmental impact. Additional measures are required, including energy-efficient hardware, efficient algorithms, and improved non-AI workload efficiency (e.g. CI/CD), alongside future work on inference-stage emissions, transfer learning, cloud versus IoT assessments, and education.

Research Questions

RQ1: How does model complexity affect energy consumption and carbon footprint during train-ing?

RQ2: Are the gains in performance from the more complex models worth the trade-offs associated with higher carbon emissions?

Methodology

Task & Data

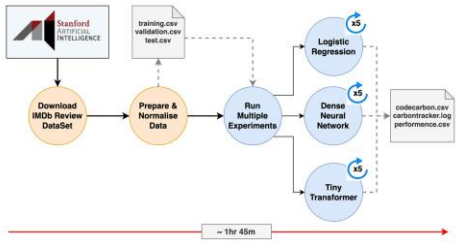
- IMDb binary sentiment classification (positive/negative), 50,000 reviews
- Train/validation/test split: **80/10/10**

Models (increasing complexity)

- Logistic Regression
- Dense Neural Network
- Tiny Transformer

Experiment Setup

- Local controlled environment: MacBook Air (Apple M3, 16GB RAM)
- **5 runs/model**; report mean values
- Track energy & emissions with **CodeCarbon** and **CarbonTracker**



Automated pipeline used to run repeatable experiments.

QR Code for Recording

A Comparative Analysis of Traditional and Serverless Kubernetes Architectures

Xinqi Pei
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218757@myTUDublin.ie

Introduction

Cloud-native systems increasingly support real-time analytics and AI workloads that demand low latency, elastic scaling, and cost efficiency. Traditional Kubernetes Pods provide predictable performance but often suffer from idle resource waste, while serverless Kubernetes platforms (e.g. Knative) introduce event-driven scaling and pay-per-use efficiency, at the cost of cold-start latency.
This research presents an empirical comparison of traditional Kubernetes Pods and serverless Kubernetes deployments using a containerised Flask-based analytics API, instrumented with Prometheus. The study evaluates latency, CPU and memory utilisation, throughput, and cost, under steady and burst traffic patterns, to identify optimal deployment strategies for real-time analytics workloads.

Sub Topic 1

Research Questions
RQ1: How does serverless Kubernetes compare to traditional Pods in request latency?
RQ2: What are the cost-efficiency differences between both models?
RQ3: How do CPU and memory utilisation patterns differ under real-time workloads?

Sub Topic 2

Test Environment
-Kubernetes (Minikube)
-Flask-based analytics API
-Prometheus & Grafana for observability
-Docker (Python 3.10-slim)

Sub Topic 3

1. Latency Performance

- Traditional Pods deliver lower & more predictable latency
- Serverless deployments experience cold-start delays
- Latency parity observed during steady traffic

2. Resource Efficiency

- Traditional Pods show significant over-provisioning
- Serverless Pods reduce idle CPU & memory by 70–80%
- Minimal memory working set under all conditions

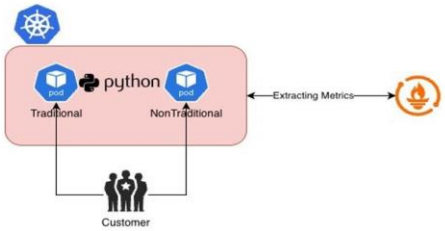
3. Cost Effectiveness

- Serverless Kubernetes yields substantially lower cost
- Best suited for bursty & intermittent workloads
- Traditional Pods remain ideal for latency-critical services

Topic Overview

This research compares traditional Kubernetes Pods and serverless Kubernetes architectures for real-time analytics workloads. Traditional Pods provide predictable performance but suffer from idle resource overhead, while serverless Kubernetes enables elastic, event-driven scaling with improved cost efficiency at the expense of potential cold-start latency. The results show that a hybrid Kubernetes–serverless approach offers the most effective balance between performance stability and resource efficiency for cloud-native analytics systems.

Figure 9: Kubernetes Application Metrics Flow to Prometheus



Conclusions and Future Work

The study shows that serverless Kubernetes can deliver comparable performance to traditional Pods for lightweight analytics workloads while significantly reducing resource waste and operational cost. Traditional deployments remain optimal for latency-critical services, whereas serverless models are better suited to dynamic traffic patterns. Future work will explore predictive scaling techniques, multi-node deployments, and evaluation on managed cloud platforms.

QR Code for Recording

An Evaluation of Kubernetes Security Mechanisms for DoS Prevention: A Comparative Analysis of Calico Network policies, K-Rail & Open Policy Agent, and Consul Rate Limiting on AWS EKS

Mia Kuric
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00218710@mytudublin.ie

Introduction

Kubernetes has become the dominant orchestration platform for cloud-native applications due to its scalability, automation, and portability. However, its widespread adoption and distributed architecture have also made it a prime target for Denial-of-Service (DoS) attacks. DoS attacks aim to exhaust system resources such as CPU, memory, or network bandwidth, leading to service degradation or complete unavailability. In Kubernetes environments, these attacks are particularly impactful because of shared cluster resources, complex service dependencies, and extensive east–west traffic between microservices. This project evaluates the effectiveness of three widely used Kubernetes security mechanisms operating at different layers of the stack: Calico Network Policies (network layer), K-Rail with Open Policy Agent (OPA) (admission and configuration layer), and Consul Rate Limiting (application layer). Controlled DoS experiments were conducted in both AWS Elastic Kubernetes Service (EKS) and Minikube environments to analyse their impact on system availability, performance, and resource consumption. The results provide practical insights into how defence-in-depth strategies can improve Kubernetes resilience against DoS attacks.

Objectives

- Evaluate the effectiveness of Kubernetes security mechanisms in mitigating Denial-of-Service (DoS) attacks
- Analyse the performance impact of Calico Network Policies, K-Rail with OPA, and Consul Rate Limiting when deployed individually
- Assess how deployment environment (AWS EKS vs. Minikube) influences DoS resilience
- Develop practical recommendations for implementing multi-layer Kubernetes security strategies that balance protection and performance

Research Question

How effective are Calico Network Policies, K-Rail & Open Policy Agent, and Consul's Rate Limiting in mitigating DoS attacks in cloud-based Kubernetes environments?

Performance Impact and Security Trade-Offs

Performance Impact
Calico network policies imposed very little additional CPU or memory usage and were effective at filtering unauthorised traffic at the network layer. However, once traffic was allowed, Calico offered no protection against application-layer attacks such as HTTP flooding.

K-Rail combined with OPA introduced almost no runtime overhead, since policy enforcement occurs during admission rather than at execution time. Although it does not actively stop live attack traffic, it helped limit the potential impact of attacks by enforcing safer configurations and resource constraints.

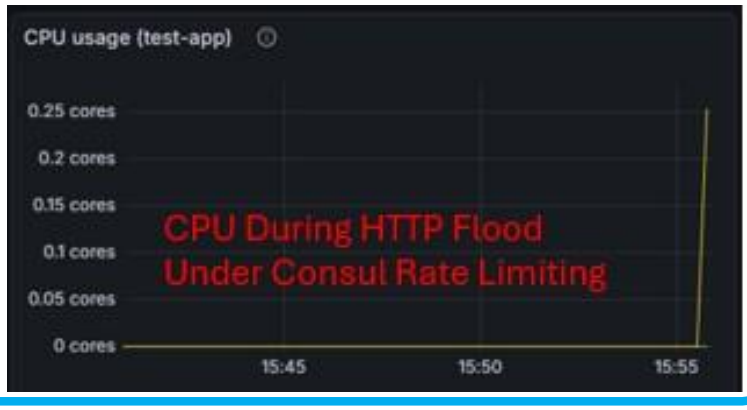
Consul rate limiting delivered the most effective defence against HTTP flood attacks. This came at the cost of increased CPU and memory consumption, mainly due to the use of Envoy sidecar proxies, which became more noticeable under sustained high request rates.

Security Trade-offs
Mechanisms with low resource overhead tend to offer protection over a narrower attack surface
Controls operating at the application layer provide stronger resistance but require greater system resources
No single solution was sufficient to fully protect against all forms of DoS attacks

Key insight: Security effectiveness and performance overhead are inherently linked; trade-offs are unavoidable.

Topic Overview

Mechanism	Effective Against	Ineffective Against
Syncookies (Linux)	Partial SYN flood mitigation	High CPU pressure
Calico	Block unauthorized L3/L4 traffic	Allowed HTTP floods
K-Rail & OPA	Prevent unsafe pod configs	Runtime DoS
Consul Rate Limiting	HTTP floods (L7)	SYN floods



Conclusions and Future Work

This study demonstrates that Denial-of-Service (DoS) resilience in Kubernetes cannot be achieved through only one security mechanism. Calico Network Policies provide efficient early packet filter-ing, K-Rail with OPA strengthens cluster security posture by preventing insecure deployments, and Consul Rate Limiting delivers robust application-layer protection. When combined, these mecha-nisms form a defence-in-depth strategy that significantly improves availability and stability under DoS conditions.

Future research should extend this work by:
- Evaluating additional service meshes such as Istio or Linkerd
- Testing larger Kubernetes clusters
- Investigating adaptive, telemetry-driven DoS mitigation using eBPF

QR Code for Recording

A Comparative Study of Flyte and Kubeflow for Orchestrating Model Retraining Pipelines

Magesh Nandikkara
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00218707@mytudublin.ie

Machine Learning Operations (MLOps) has emerged as a critical discipline for managing the lifecycle of ML models. MLOps has to address model drift, the degradation of model performance over time due to changes in data. Continuous training is the primary strategy to address model drift. This research presents a comparative study of two prominent Kubernetes-native orchestration platforms, Flyte and Kubeflow. The research will primarily focus on their efficacy in implementing adaptive model retraining pipelines. The study's core contribution is a quantitative and qualitative evaluation. In the quantitative analysis, the platforms are benchmarked across key performance metrics, including pipeline execution time, CPU and memory utilization, and scalability under increased data load. In the qualitative analysis, crucial operational characteristics and the developer experience are assessed. This research addresses two primary questions: (1) How can Flyte's dynamic workflow be effectively utilized for adaptive retraining logic? and (2) How does Flyte compare to Kubeflow across performance and operational metrics for this specific use case? The findings provide empirical data and insights to guide MLOps practitioners in making informed decisions when selecting an ML workflow orchestration tool.

Answering Research Question 1

The @dynamic workflow proved to be highly effective for implementing the adaptive retraining logic. The ability to generate a Directed Acyclic Graph (DAG) at runtime allowed conditional logic to be expressed in a Pythonic statement. This was both intuitive to write and easy to understand. The desired adaptive behaviour was correctly implemented with minimal code complexity and it did not appear to introduce any significant performance overhead. It rep-resents a powerful architectural pattern which is well-suited for complex, decision-driven work-flows where the structure of the DAG cannot be fully known at compile time.

Answering Research Question 2

Flyte is superior when it comes to developing and executing an adaptive model retraining pipeline. In Flyte, a brand new DAG is created at runtime while in KPF, the DAG is largely static, and "dynamism" can be achieved through specific control flow operators such as `dsl.ParallelFor`. Flyte's @dynamic is superior to KFP if massive parallelism is needed because Flyte handles aggregation natively. In terms of overall performance, Flyte was faster and more resource efficient compared to KFP. In terms of operational characteristics and developer experience, Flyte provided a simpler, more familiar, and more productive development process. Its Python native authoring, strong typing, local testing, and high-level abstractions significantly

Experimental Design

Flyte single cluster (using `flyte-binary`) and Kubeflow Pipelines (KFP) standalone deployment paths were selected for the comparison. The Kube Prometheus Stack was chosen for monitoring because it simplified the process of setting up monitoring for Kubernetes. The `flyte-binary` was deployed using a modified Helm chart on a single node Amazon Elastic Kubernetes Service (EKS) cluster along with the Kube Prometheus Stack. KFP was installed on a single node Google Kubernetes Engine (GKE) cluster via a one-click deployment route. The Kube Prometheus Stack was once again installed using Helm on this GKE cluster. Two synthetic, binary classification datasets were designed: `train_data` and `drifted_data`. A Logistic Regression model was first trained on the `train_data` dataset. The model was then exposed to `drifted_data` which was different from `train_data`. This simulated drift and the model performance degraded which triggered model retraining. To check scalability of the ML pipeline, `train_data` and `drifted_data` were augmented by three times to 15 rows each. Logistic Regression model was chosen for the retraining pipeline because it trains incredibly fast which is perfect for repeated pipeline executions. The retraining pipeline in Flyte was implemented using the `flytekit` SDK. In the base use case, the pipeline was designed to train a model, measure the model accuracy, introduce drift, detect the drift, retrain the model on the drifted data, tune the model with a default hyperparameter value of one and return model accuracy. The base case pipeline was executed with five hyperparameter values to understand how the pipelines work during hyperparameter tuning. Prometheus was configured to scrape utilisation metrics from all pods. The experimental design is shown in Figure 1.

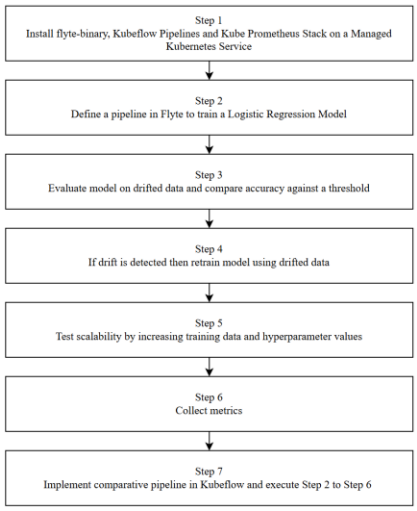


Figure 1: Overview of the experimental design.

Experiment Findings

Metric	Flyte (Mean ± Std Dev)	Kubeflow (Mean ± Std Dev)
Total Pipeline Time (s)	118.3 ± 1.79	196.4 ± 3.14
Retraining and Tuning Time (s)	52.7 ± 1.10	16.6 ± 1.62
Peak Total Memory (GiB)	0.370 ± 0.03	1.988 ± 0.05
Average Total Memory (GiB)	0.247 ± 0.02	1.849 ± 0.02
Peak Total CPU (mCPU)	10.79 ± 0.55	313.5 ± 49.85
Average Total CPU (mCPU)	7.07 ± 0.38	140.5 ± 22.33
Reliability (Success Rate %)	100%	100%

Table 1: Comparison of Flyte vs Kubeflow Base Case

Metric	Flyte (Mean ± Std Dev)	Kubeflow (Mean ± Std Dev)
Total Pipeline Time (s)	120.5 ± 1.43	190.6 ± 5.43
Retraining and Tuning Time (s)	54.2 ± 0.87	16.4 ± 0.49
Peak Total Memory (GiB)	0.38 ± 0.02	2.055 ± 0.04
Average Total Memory (GiB)	0.25 ± 0.01	1.897 ± 0.03
Peak Total CPU (mCPU)	11.05 ± 0.31	313.8 ± 27.81
Average Total CPU (mCPU)	7.277 ± 0.21	126.67 ± 19.51
Reliability (Success Rate %)	100%	100%

Table 2: Comparison of Flyte vs Kubeflow Scaling Case

Metric	Flyte (Mean ± Std Dev)	Kubeflow (Mean ± Std Dev)
Total Pipeline Time (s)	127.8 ± 0.87	192.5 ± 5.18
Retraining and Tuning Time (s)	62.6 ± 0.80	16.2 ± 0.40
Peak Total Memory (GiB)	0.51 ± 0.04	2.103 ± 0.03
Average Total Memory (GiB)	0.34 ± 0.02	1.937 ± 0.02
Peak Total CPU (mCPU)	14.94 ± 0.63	371.2 ± 39.02
Average Total CPU (mCPU)	9.832 ± 0.41	142.8 ± 12.88
Reliability (Success Rate %)	100%	100%

Table 3: Comparison of Flyte vs Kubeflow Hyperparameter Tuning

Conclusions and Future Work

Flyte consistently outperformed Kubeflow in quantitative benchmarks. The qualitative analysis revealed that Flyte has a substantial advantage over Kubeflow in terms of developer experience. However, for large organisations looking to implement a highly customisable and all-encompassing ML platform, Kubeflow remains a powerful option. The research highlighted that the "best" MLOps platform ultimately depends on the context.

Future work should focus on expanded comparison, multi-cluster analysis, larger datasets, cost-benefit analysis, integration with other MLOps tools and sophisticated retraining triggers.

QR Code for Recording

NOTES