International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.56472/ICCSAIML25-126
Eureka Vision Publication | ICCSAIML'25-Conference Proceeding

Original Article

Hybrid AI on IBM Z: Options and Technical Insights

Chandra Mouli Yalamanchili
Independent Researcher from USA.

Abstract *- The rapid growth of AI (Artificial Intelligence) and ML (Machine Learning) over the last few years has enabled organizations to seamlessly integrate real-time decision models into their existing applications for more efficient processing. In response to this growth in AI, IBM has increasingly built upon its mainframe architecture most notably IBM Z by incorporating newer technologies like support for Python, containerized workloads through zCX, and virtualization layers like z/VM and zKVM. The introduction of the Telum processor, with built-in AI acceleration, further positions IBM Z as a strong candidate for running AI workloads right where the data resides. This paper explores several options that can be used to deploy ML model training and inference on IBM Z's ecosystem. It highlights how IBM Z's performance, security, and co-location with enterprise data make it an ideal environment for hybrid AI workloads. This paper takes an example use case of real-time credit card fraud detection and explores how predictive models can be deployed on IBM Z using Python. This paper also explores the full transaction flow to reflect the integration between existing COBOL, HLASM, or Java applications and how the Python-based fraud scoring service would work in practice.*

Keywords *- Hybrid AI architecture; IBM Mainframe; IBM Z; Predictive Modeling; Fraud detection; Python on IBM Mainframe; zCX; Telum; z/OS; z/VM; zKVM; ONNX; PMML; MLz; Cloud Pack for Data.*

1. Introduction

AI (Artificial Intelligence) and ML (Machine Learning) have revolutionized the enterprise computing environment over the last few years. What started as data science exploration has grown into the production of real-time fraud detection, transaction analysis, and automated decision-making at scale. As organizations operate in today's high-stakes environments, where milliseconds translate to millions, they are more concerned with moving AI closer to their data for governance and performance. The IBM Z platform is known for its high reliability, scalability, and security in transactional processing, and it has evolved over the years to support modern AI and ML demands on the IBM Z platform. In the earlier days, running ML models often meant offloading mainframe data to external platforms for processing. While operational, this approach introduced latency in mission-critical workloads, increased overall architecture complexity, increased cost due to additional distributed infrastructure, and raised compliance concerns around data movement.

Recognizing these limitations, IBM introduced a series of enhancements to make AI more native to the IBM Z environment:

- IBM z13 and z14 systems added support for embedded analytics and leveraged zIIP (z Integrated Information Processor) processors to offload eligible workloads efficiently.
- IBM z15 extended the platform's capabilities with stronger containerization support and built-in data privacy and encryption features at scale.
- Most notably, in 2021, IBM announced the Telum processor, which introduced on-chip AI acceleration. For the first time, inferencing tasks could be executed directly within the CPU pipeline, allowing AI models to run alongside transactional workloads without leaving the platform. [1]

With Telum and the supporting software stack, IBM Z now enables low-latency, real-time inference natively, eliminating the need to ship data elsewhere and making it an ideal environment for high-volume, time-sensitive use cases such as credit card fraud detection.

This paper explores how a hybrid AI architecture can integrate predictive modeling into IBM Z systems. We walk through the full machine learning lifecycle: training models in virtualized environments such as z/VM and zKVM and deploying them through Python-based RESTful services hosted on z/OS UNIX or within zCX containers. Using a unified use case real-time fraud detection we demonstrate how AI and traditional workloads coexist on the mainframe, delivering insight and action with minimal latency and maximum control.

2. Benefits of Building, Training, and Running Machine Learning Models on IBM Z Hardware

IBM Z hardware offers several unique advantages that position it as a highly effective platform for executing machine learning workloads across the entire model lifecycle:

- **Proximity to Enterprise Data**: Most enterprise data already reside on IBM Z systems in databases like Db2, VSAM datasets, or transactional logs such as SMF. Running ML workloads close to this data reduces the need for costly and complex data movement, which is necessary for real-time applications like fraud detection and anomaly analysis. [1][6]
- **Security and Data Governance**: IBM Z is well known for its state-of-the-art security features, including pervasive encryption and compliance-focused access controls. These features of IBM Z enable organizations to avoid exporting data to less secure environments by implementing training and inference directly on the IBM Z secure environment, thereby ensuring regulatory compliance with standards like GDPR and HIPAA. [6]
- **Performance and Scalability**: With hundreds of cores, massive I/O throughput, and massive memory bandwidth, IBM Z provides the performance to run ML pipelines at scale. The newly introduced Telum processor has on-chip AI inferencing capabilities allowing inference to happen alongside transaction processing and reducing scoring latency from milliseconds to microseconds. [1]
- **Resilience and Availability**: IBM Z systems are built for high availability (99.999%), making them ideal for mission-critical AI scenarios without downtime. This resiliency ensures smooth training schedules and frequent model deployments.
- **Flexible Execution Environments**: As with any other implementation, IBM Z provides a range of environments like running Python on z/OS UNIX, deploying containers using zCX or leveraging more powerful ML platforms like Cloud Pak for Data (CP4D) on Z, and many more options that provide flexibility to data scientists and ML engineers to choose any option that's more suitable for their existing ecosystem. [1][2][3]
- **Integrated AI Tooling**: The integration with Watson Machine Learning, AutoAI, and OpenShift-based orchestration offers an enterprise-ready path to manage models, track their lineage, perform versioning, and automate deployment in a CI/CD pipeline.
- **Real-Time Inference Potential**: For use cases like credit card fraud detection, IBM Z's combination of low-latency transaction processing and native inferencing support with Telum chips creates a seamless environment to detect and respond to fraud as transactions occur. [4]

These benefits collectively make IBM Z a viable platform for machine learning and a strategic one for enterprises looking to leverage AI without compromising security, performance, or compliance.

3. ML Model Inference Options on IBM Z

While the training of machine learning models can be performed across a range of environments including public cloud platforms and virtualized Linux systems the location of inference execution remains critical, particularly when the model's output is needed to support core IBM Z workloads. Offloading data to off-host scoring engines as part of real-time transaction processing applications introduces latency and potential compliance risk due to data transmission. Instead, having inference close to the transactional environment is preferred to ensure performance and data security. IBM Z offers various options for executing on-host inference workloads depending on model complexity, desired runtime environment, and operating system integration requirement

Below are some of the prominent approaches:

- **z/OS UNIX (Python on USS - z/OS UNIX System Services):** This lightweight and highly flexible approach allows Python scripts to run directly within the z/OS UNIX System Services (USS) environment. Using Rocket Python or Anaconda distributions, this option works well for implementing simple models and REST-style inference APIs.
- **z/OS Container Extensions (zCX):** For applications currently benefitting containerized development, zCX enables Docker-compatible model containers built with frameworks such as Flask or FastAPI to run directly within z/OS. This option provides an isolated and containerized environment for model inference while integrating seamlessly into REST-based enterprise workflows using same-host connectivity. [2]
- **z/VM with Linux (Ubuntu or RHEL):** This configuration supports the execution of full ML stacks, including TensorFlow and PyTorch, making it a robust choice for training and inference. This option offers greater flexibility to organizations wanting to perform all model development, training, and deployment on the IBM Z platform.
- **zKVM (KVM for IBM Z):** zKVM provides lighter Linux virtualization with lower overhead compared to z/VM. This option is suitable for running inference microservices or model deployment in light environments.
- **IBM Machine Learning for z/OS (MLz):** MLz offers a structured, enterprise-grade solution for model deployment and scoring on z/OS. With REST APIs, model versioning, and lifecycle management features, MLz is ideal for teams requiring governance, auditability, and integration with enterprise DevOps pipelines [6].

- **Liberty JVM (within or outside CICS):** Java applications can support inference logic via frameworks like DeepLearning4J within the Liberty runtime, deployed within CICS, or as a started task JVM. The strategy is beneficial for organizations that are adopting the Java platform.
- **Cloud Pak for Data on Z:** As a full-featured platform built on OpenShift, Cloud Pak for Data provides end-to-end support for the ML lifecycle including model training, deployment, monitoring, and governance. While resource-intensive, it offers unmatched completeness for enterprise AI pipelines [7].
- **ONNX (Open Neural Network Exchange) on IBM Z:** ONNX is an open standard for representing machine learning models across frameworks. IBM Z supports ONNX model execution by converting them into optimized C code using IBM's Deep Learning Compiler (ONNX-DLC) or GitHub-supported tools [8]. This method eliminates the need for Python or Java runtimes at inference time, offering high performance with minimal overhead.
- **PMML Conversion:** Predictive Model Markup Language (PMML) is an appropriate option for simpler models. PMML-encoded models can be executed within COBOL, Java, or C environments on IBM Z. It is a low-friction and accessible solution best suited for legacy modernization strategies [8]

4. Comparison of Inference Deployment options on IBM Z

Table 1. Comparison of different options for deploying ML models on IBM Z.

Option	Runtime Type	Language(s)	Setup Complexity	Suitable for	Real-Time Capable	On-chip AI Support	Notes
z/OS UNIX (Python)	Script-based	Python	Low	Lightweight inference, scripting	Yes	No	Requires Rocket/Anaconda Python; great for REST APIs
zCX (Docker Containers)	Containerized	Python, REST	Medium	Microservices, RESTful APIs	Yes	No	Docker-style deployment needs TCP/IP routing config
z/VM + Linux	Full VM	Python, Java, and others.	Medium–High	Training & batch inference	Not ideal for real-time	No	Flexible, but outside the transactional context
zKVM(Linux KVM)	Lightweight VM	Python, Java, and others.	Medium	Inference microservices	Yes (with tuning)	No	Leaner than z/VM; can host Docker or RREST-based APIs
IBM MLz	Managed z/OS service	REST (model agnostic)	Medium	Structured model serving with versioning	Yes	No	Full lifecycle control; ideal for regulated environments
Liberty JVM (CICS or standalone)	Java runtime	Java	Medium	Java app integration	Yes	No	Supports DeepLearning4J; good for Java EE workloads
Cloud Pak for Data on Z	Kubernetes/OpenShift	Python, R, Java, and others.	High	Full lifecycle incl. training	Yes	No	Enterprise-grade MLOps; good for hybrid pipelines
ONNC (ONNX to C)	Compiled native	C	Medium–High	Lightweight, fast scoring	Yes	Yes (via Telum)	Requires ONNX model; minimal runtime dependencies
ONNX (Compiled to C)	Compiled native	C	Medium - High	Lightweight, fast scoring	Yes	Yes (Telum-optimized	Converts ONNX to C for native inference using

						)	IBM DLC or GitHub tools
PMML	Transpiled / interpreted	COBOL, Java, C	Medium	Legacy integration	Yes (if wrapped)	No	Great for simple models, especially in COBOL/Java

5. Use Case – Credit Card Fraud Detection

Credit card fraud detection is a critical use case where milliseconds matter. [5] Every transaction must be evaluated in real-time during an authorization request originating from a point-of-sale, ATM, website, or mobile app. The system must detect fraudulent activity without delaying the transaction response to the originating source. However, the underlying fraud detection system must evaluate several factors like behavioral patterns, risk factors, and transaction parameters to make this decision all within a few milliseconds. ML can help in this scenario by making this decision faster while avoiding the need for complex rules-based systems currently used for fraud detection.

5.1 The Need for On-Platform AI

Running fraud detection models directly on IBM Z can significantly reduce the latency added to assess risk. Traditional approaches often require data movement to external scoring engines (cloud or distributed), introducing latency and increasing the risk of data exposure. We eliminate that overhead by bringing inference closer to the transaction processing system. The proximity inference solution makes IBM Z where most financial institutions already run authorization flows an ideal platform for embedding ML scoring.

As highlighted in reference [4], fraud detection models commonly leverage features such as:

- Historical transaction frequency per merchant category.
- Spending pattern deviations.
- Location changes.
- Time-of-day patterns.
- Device or card-present behavior.

These behavioral vectors are generally built on established cardholder profiles and are best maintained and stored in IBM Z databases or VSAM files. Inference execution on IBM Z allows such features to be computed and scored based on established workflows without introducing extrinsic dependencies or redundancy data storage.

5.2 Model Lifecycle: Traditional vs. IBM Z

Below is a conceptual comparison between traditional and IBM Z-native deployment models for fraud detection scoring.

5.2.1 Traditional Architecture (depicted in Figure 1):

- Transaction arrives in CICS/IMS.
- Metadata is sent over the network to the external scoring engine.
- External API executes inference.
- The risk score is returned to the mainframe app.
- Decision logic is executed.

Below are some challenges with this approach:

- Added latency (network round trip).
- Potential security/compliance risk due to data movement.
- Dependency on distributed AI infrastructure.
- Doesn’t take benefit of AI-acceleration via Telum chip.

5.2.2 IBM Z-Native Architecture (e.g., z/OS UNIX Python or zCX) (depicted in Figure 2):

- Transaction arrives in CICS/IMS.
- CICS invokes local REST API (running on USS or in zCX).
- The model inference is executed on-platform.
- The score is returned immediately.

- Decision logic continues without leaving LPAR (Logical Partition).

Below are some benefits with this approach:

- Minimal latency (no network hops).
- Simplified architecture.
- Easier audit/compliance.
- Co-location with data, code, and business logic.
- AI-acceleration through Telum chip.

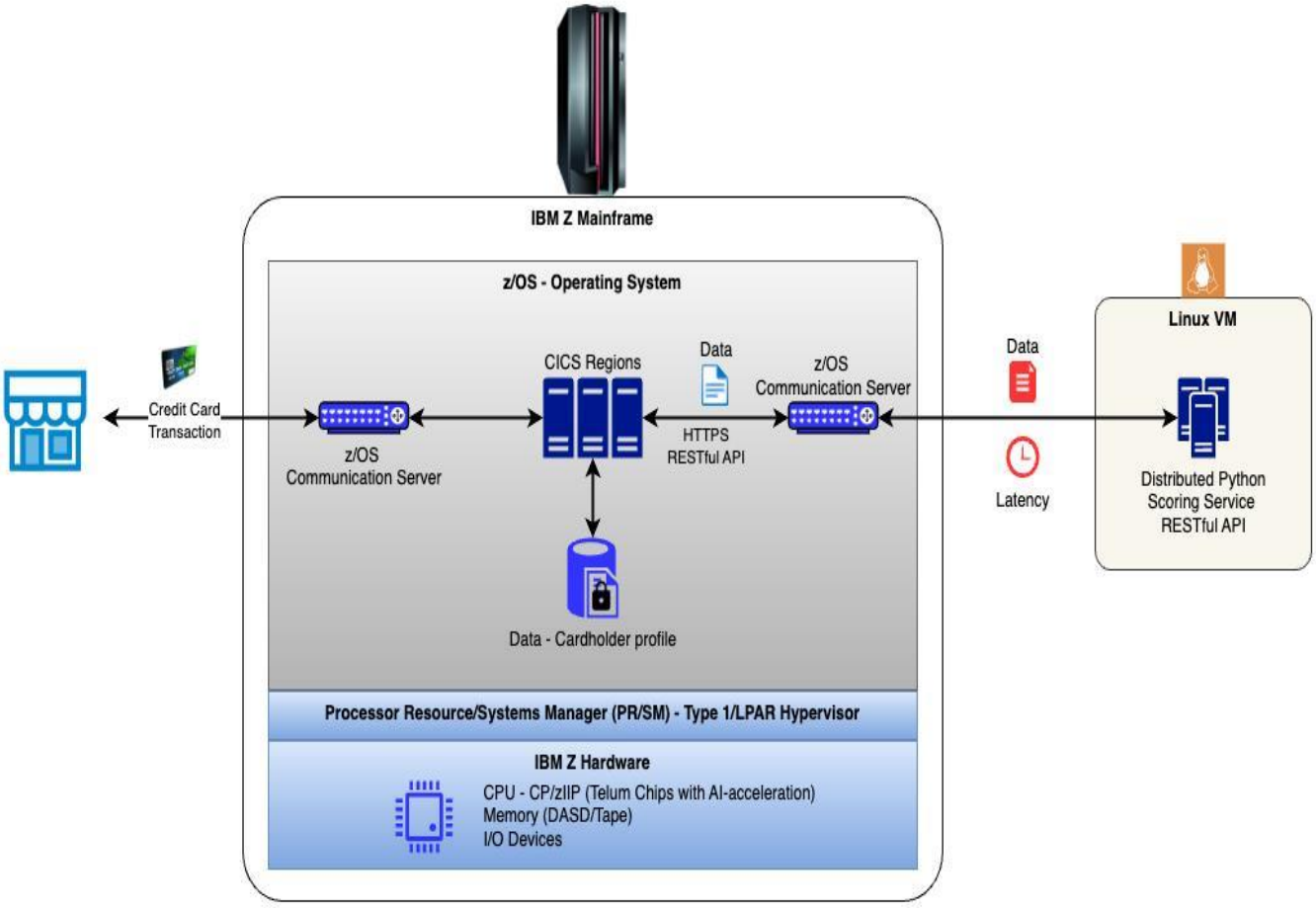

Figure 1. Traditional transaction fraud scoring with off-host inference

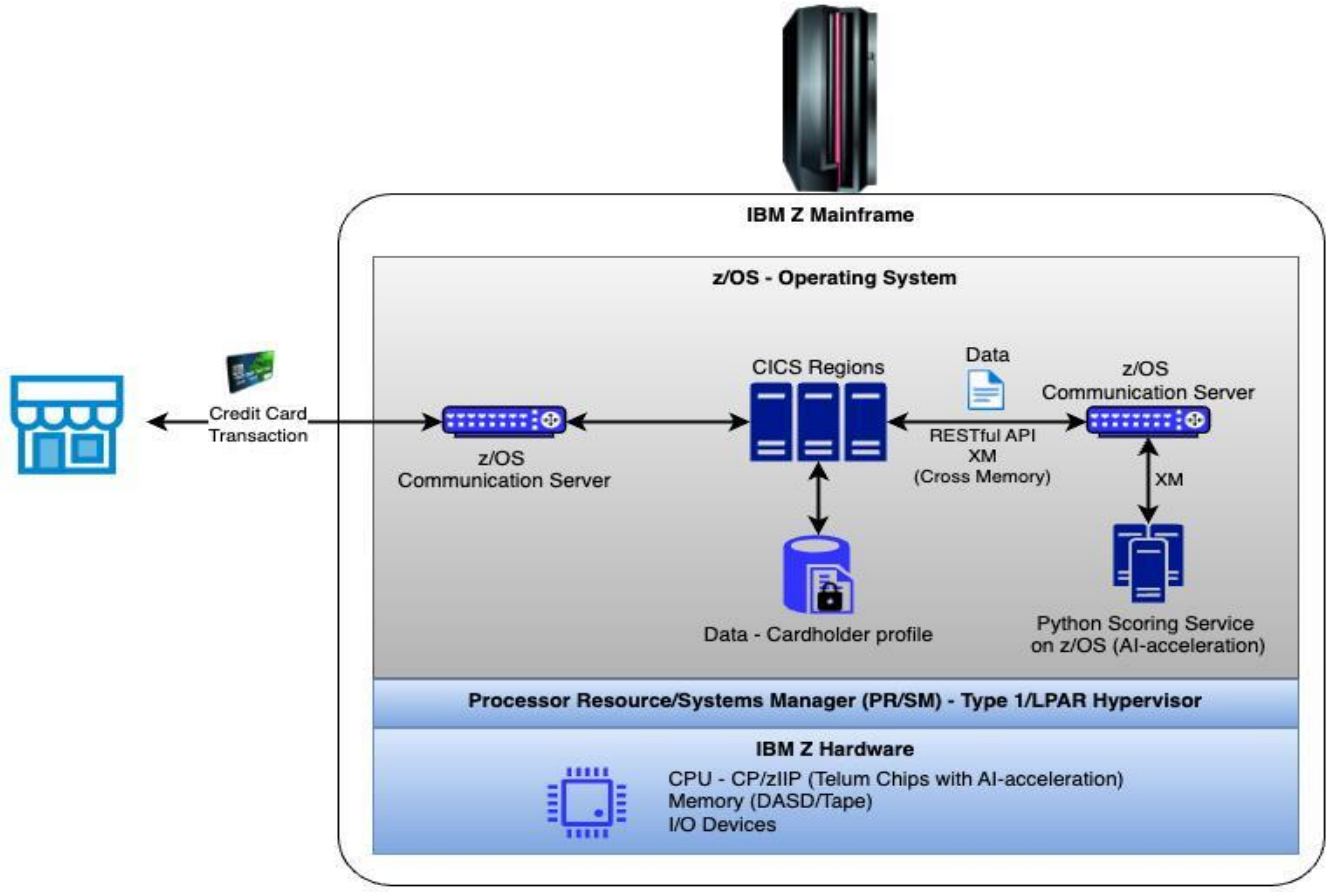

Figure 1. IBM Z native transaction fraud scoring using USS-based Python inference taking benefit of same-host XM (cross-memory) call from CICS as well as AI-acceleration from Telum chip

5.2.3 Model Inference through multiple options on IBM Z

Below is how the same fraud detection use case could be executed using different IBM Z inference mechanisms:

- **Python on z/OS UNIX:** Python script loads the model, accepts a REST call, and returns a fraud score. CICS or Java applications running on IBM Z invoke the Python inference service during transaction processing using EZA sockets or HTTP clients. Feature data is passed in JSON.
- **zCX (IBM z/OS Container Extensions) Container:** The fraud scoring service is containerized using Flask or FastAPI. The CICS app routes transaction metadata via REST to the zCX endpoint. zCX performs inference and responds.
- **MLz Service:** The model is deployed and versioned inside MLz. z/OS app invokes a model scoring endpoint via REST. MLz handles lifecycle management and audit logging.
- **z/VM or zKVM (Linux):** Training may occur here. If inference is hosted here as well, it requires network configuration to take benefit of HiperSockets and reduce latency. REST API on Linux guest accepts scoring requests from z/OS.
- **ONNX → C with DLC:** The pre-trained ONNX model is compiled into a C program via DLC. This C executable can be invoked by z/OS batch or COBOL as a callable program, allowing extremely fast scoring with minimal dependencies. [8]
- **PMML Model:** A simple model is exported in PMML format and embedded in a COBOL or Java scoring engine. No external runtime is required; scoring occurs in line with application logic. [8]

6. Sample code depicting the z/OS UNIX-based Python application to implement the Fraud detection use case

This section demonstrates how machine learning inference can be executed directly on IBM Z; this section also outlines a working example that runs a model scoring service on z/OS UNIX (USS) using Python. CICS or batch applications written in COBOL or Java can invoke this REST API for real-time fraud detection scoring. This example assumes that the model has been

pre-trained (either off-platform or on a z/VM environment) and exported as a serialized .pkl file. The model is then deployed to the USS file system alongside a lightweight Python inference service built with Flask.

6.1 Python Fraud scoring API service (app.py)

```
from flask import Flask, request, jsonify
import joblib

app = Flask(__name__)
model = joblib.load("fraud_model.pkl")

@app.route("/score", methods=["POST"])
def score():
    data = request.get_json()
    features = data["features"]  # list of features: [txn_amt, time_delta, ...]
    prediction = model.predict([features])
    score = prediction[0]
    return jsonify({"fraud_score": int(score)})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

This API listens on port 5000 and expects JSON input with a features array. It returns a fraud score as a simple integer response.

6.2 JCL to run the Python service as the started task

```
//PYAPI   JOB (ACCT),' START PYTHON API',CLASS=A,MSGCLASS=X
//STEP1   EXEC PGM=BPXBATCH
//STDPARM DD *
SH cd /u/youruser/fraud-api && python3 app.py
```

6.2.1 JCL Details:

- BPXBATCH runs Python under USS.
- Ensure that Rocket Python or Anaconda is installed and Python3 is available in PATH.
- The fraud_model.pkl and app.py files should be deployed in /u/youruser/fraud-api/.

6.3 COBOL Client that invokes the Python REST API

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. FRAUDCALL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  JSON-REQUEST    PIC X(200) VALUE
          '{"features":[125.50, 30, 2, 0, 1]}'.
       01  RESPONSE        PIC X(100).
       PROCEDURE DIVISION.
          CALL 'EZACIC02' USING
             'POST'                    *> HTTP Method
             'http://127.0.0.1:5000/score' *> URL
             JSON-REQUEST
             LENGTH OF JSON-REQUEST
             RESPONSE
             LENGTH OF RESPONSE.
          DISPLAY "Returned Score: "RESPONSE
          STOP RUN.
```

6.3.1 COBOL Program Details:

- This program uses the EZA HTTP client within CICS or batch to call the REST service.
- EZA is the standard IBM supplied TCP/IP client for CICS and batch programs.
- Java or Python-based callers can substitute this COBOL client code in Java EE or Liberty environments.

This Python-based implementation is ideal for lightweight, scriptable integration and fast prototyping. However, the same scoring logic can be migrated to other environments discussed earlier:

- Deployed as a container in zCX.
- Wrapped in Java logic for use in Liberty JVM.
- Converted to native code using ONNX-DLC.
- Packaged into MLz for full lifecycle control.

The right choice depends on the use case requirements, runtime constraints, and integration needs.

7. Conclusion

As AI and machine learning evolve, the need to execute intelligent models where the data lives and where decisions are made has never been more important. With its history of powering mission-critical workloads, IBM Z is uniquely positioned to support AI-driven transformation without sacrificing performance, compliance, or operational integrity. This paper demonstrated that inference can be brought natively into the IBM Z ecosystem using a wide range of integration options from Python scripts running on z/OS UNIX to full-service containers deployed via zCX, from compiled ONNX models accelerated by the Telum processor to enterprise-managed scoring services like MLz. Each option has its own strengths, and the flexibility offered allows organizations to adopt the one that best fits their application architecture, team skills, and deployment goals. By applying this architecture to a real-world use case credit card fraud detection, we showcased how low-latency, high-frequency transactional environments can benefit from embedded intelligence without adding complexity or offloading sensitive data.

More importantly, this paper highlighted that IBM Z is no longer a system that hosts applications it can now reason, evaluate, and decide. As IBM pushes the boundaries with initiatives like AI on Z, Telum-enabled hardware acceleration, and further integration with open-source machine learning environments the ability to infuse conventional enterprise systems with intelligence grows. To data science professionals and enterprise architects, the message is one of certainty: There is no longer any necessity to trade-off between rich, current artificial intelligence capability or tried-and-tested stability of existing infrastructure. With IBM Z, both are within reach securely, at scale, and without compromise.

References

[1] IBM Research, " IBM’s newest chip is more than meets the AI", IBM Research, Aug 2021. [Online]. Available: https://research.ibm.com/blog/telum-processor. [Accessed: April 2025].

[2] M. Yalamanchili, "Containerization on z/OS: Running Applications with z/OS Container Extensions (zCX)", Journal of Mathematical & Computer Applications, vol. 3, no. 6, pp. 1-4, Dec. 2024. doi:10.47363/jmca/2024(3)197. [Online]. Available: https://onlinescientificresearch.com/articles/containerization-on-zos-running-applications-with-zos-container-extensions-zcx.pdf.

[3] M. Yalamanchili, "Running Linux on IBM Z: Hybrid Workloads and Cloud-Native Application Support", International Journal of Leading Research Publication, vol. 5, no. 11, Nov. 2024. doi:10.5281/zenodo.14785944. [Online]. Available: https://www.ijlrp.com/papers/2024/11/1270.pdf.

[4] M. Yalamanchili, "Credit Card Fraud Detection Using Data Science," J. Artif. Intell. & Cloud Comp., vol. 2, no. 1, pp. 1-3, 2023. doi:10.47363/JAICC/2023(2)E232.

[5] IBM Technology, " Mainframe Z and AI", YouTube, Sep. 27, 2023. [Online]. Available: https://www.youtube.com/watch?v=OSRXo56R5Ts&t=280s. [Accessed: April 2025].

[6] IBM, "Machine Learning for z/OS". [Online]. Available: https://www.ibm.com/products/machine-learning-for-zos. [Accessed: April 2025].

[7] IBM Redbooks, "Cloud Pak for Data on IBM Z", REDP-5695-00, https://www.redbooks.ibm.com/redpapers/pdfs/redp5695.pdf. [Accessed: April 2025].

[8] IBM GitHub, "Getting started with AI on IBM Z and LinuxONE systems". [Online]. Available: https://ibm.github.io/ai-on-z-101/. [Accessed: April 2025].

[9] IBM GitHub, "AI on z/OS: Solution Template for Fraud Detection". [Online]. Available: https://github.com/ambitus/aionz-st-fraud-detection/blob/main/ai_solution_template_fraud.pdf. [Accessed: April 2025].