# SmartCitySystem: Um Gêmeo Digital para Simulação Urbana e Análise de Dados de Mineração

Aluno(a): Andrei Inoue Hirata

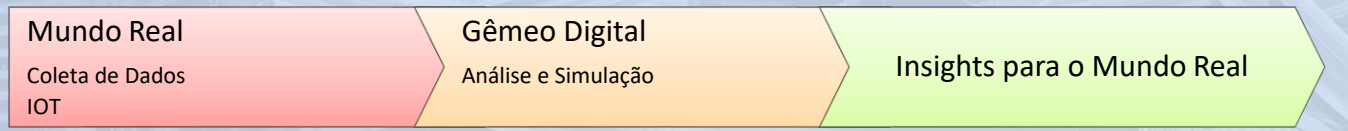
Prof(a): Veronica Oliveira de Carvalho

# Resumo

- Integração de Simulação 3D (Unity),
- Geração de dados
- Importação de Dados Reais (INMET)
- Modelos de Machine Learning (Python/Node.js)

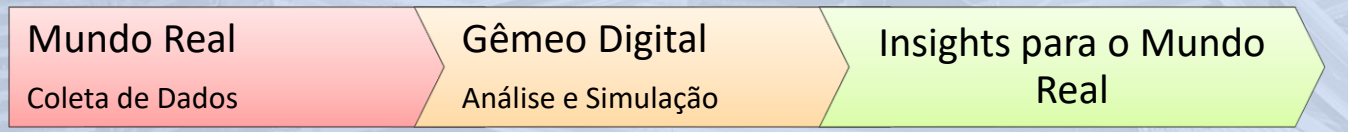
# Introdução e Contexto (a.1)

- **Definição:**
- Gêmeo Digital (Digital Twin)
  - é uma réplica virtual de um sistema físico (um objeto, um processo ou, neste caso, uma cidade).
- IoT (Internet das Coisas)
  - é a rede de objetos físicos conectados à internet que podem coletar e transmitir dados, permitindo a comunicação entre dispositivos, a automação de tarefas e a análise de informações como câmeras de tráfego, estações meteorológicas, medidores de poluição) que coletam dados.
- Big Data
  - conjuntos de dados muito grandes, complexos e de crescimento rápido que não podem ser gerenciados por softwares tradicionais de processamento de dados, como planilhas ou bancos de dados relacionais



# Introdução e Contexto (a.1)

- **Por que Cidades?**
  - Cidades são os sistemas mais complexos que existem. Elas geram um volume massivo de dados em alta velocidade (Big Data) a partir de milhões de sensores interconectados (IoT), como câmeras, estações climáticas e medidores de poluição.
- **Motivação**
  - Poder testar cenários, prever problemas (como congestionamentos ou picos de poluição) e planejar mudanças em um ambiente virtual, sem custos ou riscos do mundo real.



# Problema e Objetivos (a.1)

- **Problema:** Como integrar, em uma única plataforma, uma simulação 3D interativa (visual) com fontes de dados heterogêneas (climáticos) e modelos preditivos de ML (tráfego e poluição)?

# Problema e Objetivos (a.1)

- **Objetivo Geral:** Desenvolver o "SmartCitySystem", uma plataforma que simula cidades inteligentes
- **Objetivos Específicos:**
  - Implementar um replay de dados climáticos históricos (INMET) sob-demanda.
  - (Atividade 3) Aplicar Mineração de Dados para Detecção de Outliers no tráfego (usando LOF).
  - (Atividade 2 & 4) Aplicar Mineração de Dados para Classificação da Qualidade do Ar (usando RandomForest e SMOTE).

# Trabalhos Relacionados (a.2)

- **Simulação 3D e Gêmeos Digitais:** Muitos projetos usam Unity para simulação (Unity, 2025; Espona, 2023), mas historicamente focam na indústria (robótica, manufatura) e não na integração de dados ambientais urbanos em tempo real.
- **Análise de Dados de Tráfego:** (Cruz et al., 2017) aplicam algoritmos como o LOF (Local Outlier Factor) para detectar anomalias em dados de tráfego, mas geralmente em análises *offline*, sem uma visualização 3D integrada.
- **Análise de Qualidade do Ar:** (Vianna Jr. et al., 2025) usam ML para classificar dados da CETESB, e outros (Della-Justina, 2023) focam especificamente no desafio de **dados desbalanceados** (como fizemos com o SMOTE).
- **Esse projeto:** A literatura carece de projetos que unam essas *três* frentes: (1) simulação 3D interativa, (2) detecção de outlier em tempo real e (3) classificação de qualidade do ar em um único sistema (Unity + Node.js + Python).

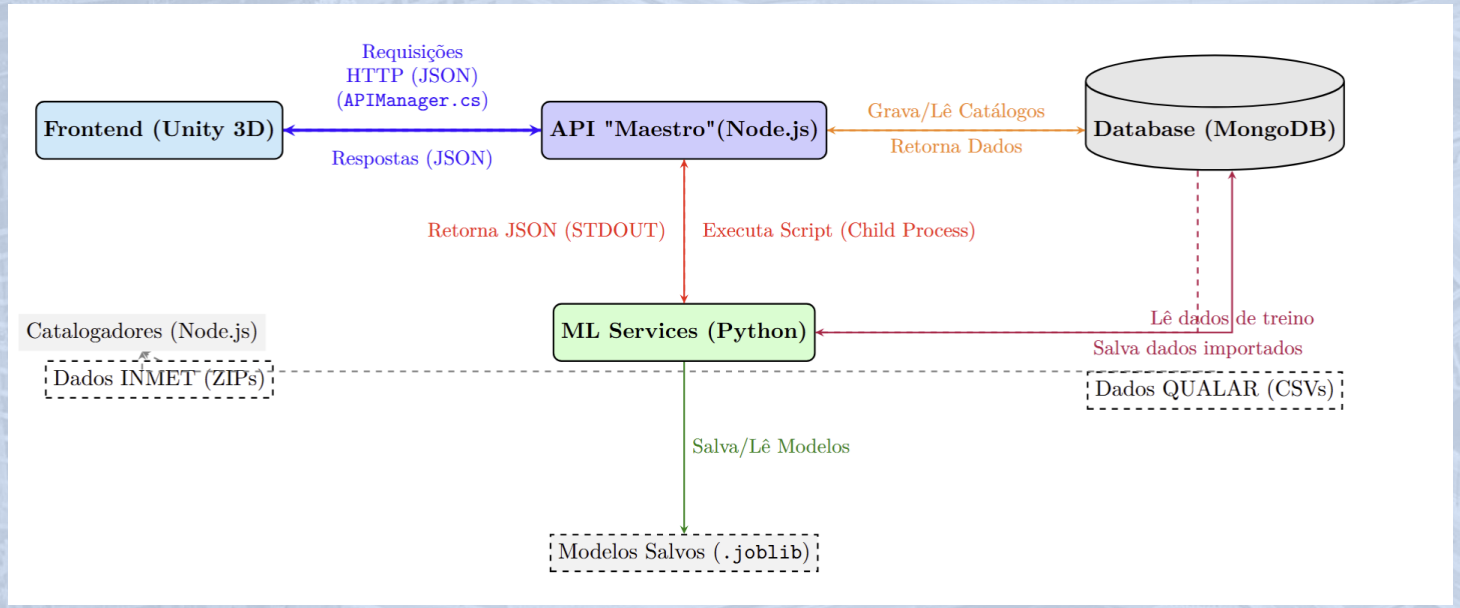
# Fundamentos (a.3) - Conceitos Chave

- **Arquitetura Cliente-Servidor:**
  - **Cliente (Frontend):** Unity 3D (para visualização e interação).
  - **Servidor (Backend):** Node.js (API "Maestro" que gerencia as requisições).
- **Microsserviços de ML via child_process:**
  - Em vez de um servidor Flask/Python separado, o Node.js executa scripts Python (.py) sob-demanda.
  - Vantagem: Mais leve, mais simples de hospedar (apenas 1 servidor) e eficiente.
- **Banco de Dados:** MongoDB (Banco NoSQL ideal para armazenar os dados JSON).

# Fundamentos (a.3) - Mineração de Dados

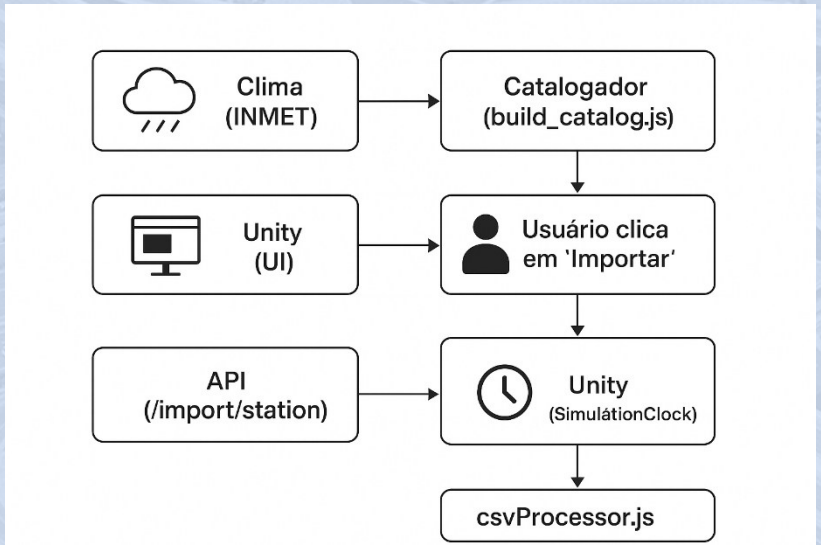
- **Atividade 3: Detecção de Outliers (LOF):**
  - Um algoritmo que encontra anomalias.
  - Ele é excelente em encontrar "outliers de densidade". Ex: Um carro parado (avgTimeOnSensor alto) em um tráfego que flui (vehicleCount baixo) é uma anomalia de densidade.
- **Atividade 2: Classificação (Random Forest):**
  - Um modelo que aprende a prever uma classe (ex: "Boa", "Moderada", "Alerta").
- **Atividade 4: Dados Desbalanceados (SMOTE):**
  - O problema de ter poucos exemplos de classes raras (ex: 100 dias "Bons" para cada 1 dia "Alerta").
  - Ele cria "exemplos sintéticos" das classes raras, forçando o modelo a aprender a detectá-las.

# Metodologia (a.4) - Arquitetura do Sistema



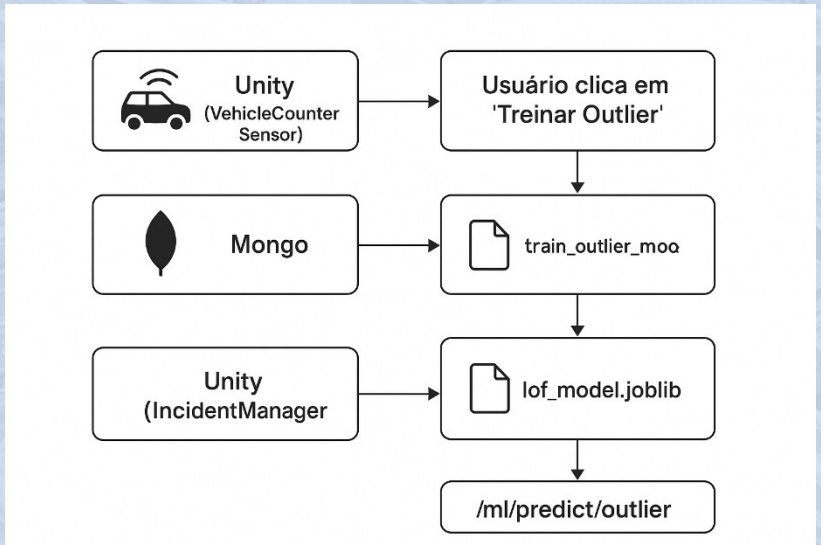
# Metodologia (a.4) - Pipeline de Dados

- Pipeline 1: Clima (INMET)
  - Catalogador (*build_catalog.js*) lê ZIPs e salva no Mongo.
  - Unity (UI) lê o catálogo via API (*/filters/years...*).
  - Usuário clica em "Importar".
  - API (*/import/station*) baixa o ZIP, processa o CSV (csvProcessor.js) e salva os dados no Mongo.
  - Unity (SimulationClock) lê os dados do Mongo via API (*/replay*) e muda o clima.



# Metodologia (a.4) - Pipeline de Dados

- Pipeline 2: Outlier (Tráfego)
  - Unity (VehicleCounterSensor) gera dados (Contagem, Tempo) e salva no Mongo.
  - Usuário clica em "Treinar Outlier".
  - API (*/ml/train/outlier*) chama o train_outlier_model.py.
  - Script Python treina (LOF) e salva o lof_model.joblib.
  - Unity (Sensor) chama a API (*/ml/predict/outlier*) em tempo real.
  - API chama predict_outlier.py (que usa o .joblib) e retorna a predição.
  - Unity (IncidentManager) instancia o "prefab" de incidente.



# Metodologia (a.4) - Pipeline de Dados

- Pipeline 3: Qualidade do Ar (Qualar)
  - (Idêntico ao Pipeline 2, mas usando qualar_importer.py e os modelos de Classificação/SMOTE)

## Experimentos e Resultados (a.5) - Recurso 1: Replay de Clima

- Desafio: Como lidar com gigabytes de arquivos CSV do INMET sem travar o servidor ou a Unity?
- Solução (Catálogo Sob-Demanda): Implementamos um catalogador que pré-processa apenas os nomes dos arquivos. A UI filtra instantaneamente.
- Resultado: A API importa apenas o arquivo CSV necessário, sob-demanda, quando o usuário clica. O SimulationClock então lê esses dados importados e ajusta o ambiente (chuva, céu)

# Experimentos e Resultados (a.5) - Recurso 2: Detecção de Outlier

- **Objetivo:** Detectar anomalias de tráfego (congestionamentos) em tempo real.
- **Experimento:**
  - Sensores 3D na Unity (VehicleCounterSensor) coletaram dados de tráfego "normal" dos carros-robô.
  - Treinamos o modelo LOF (Local Outlier Factor) com esses dados.
  - API de predição foi ativada.
  - **Teste de Anomalia:** Forçamos um carro-robô a parar dentro do sensor (desativando seu script de IA).
- **Resultado:** O sistema detectou com sucesso o "outlier" (carro parado) e instanciou um "prefab" de incidente (carro quebrado) no local.

# Experimentos e Resultados (a.5) - Recurso 3: Classificação

- Objetivo: Prever a qualidade do ar (Boa, Moderada, Alerta) baseada no contexto (hora, dia, mês).
- Desafio: A API oficial da CETESB se mostrou inacessível (bloqueando todas as requisições de filtro).
- Solução (Plano B): Implementamos um pipeline de dados local. Criamos um catálogo (build_qualar_catalog.js) para ler arquivos CSV locais (ex: qualar_2024_bauru_mp10.csv).
- Resultado: A UI filtra os dados locais, importa-os, e o SimulationClock chama a API de predição, exibindo a classe prevista ("Boa", "Moderada") e a confiança na UI.

# Discussão (a.5) - O Desafio dos Dados Desbalanceados

- Problema (Dados Reais): Dados de qualidade do ar eram altamente desbalanceados. (Ex: Boa: 5208 registros, vs Alerta: 149 registros).
- Solução (SMOTE): Aplicamos a técnica SMOTE, que criou amostras sintéticas das classes "Moderada" e "Alerta" até que todas tivessem 5208 registros.
- Discussão (Resultados):
  - O modelo treinado (com SMOTE) teve uma acurácia geral de 83%.
  - Insight: O F1-Score (que mede a precisão) foi excelente para "Boa" (0.91), mas muito baixo para "Alerta" (0.14).
  - Conclusão Acadêmica: Isso mostra que, mesmo com SMOTE, prever eventos raros é um desafio complexo. O modelo aprendeu a "apostar no seguro" (prever "Boa"), indicando que features adicionais (como dados de clima) seriam necessárias para melhorar a predição de alertas.

# Conclusão (a.6)

- **Trabalho Realizado:** Foi construída com sucesso uma plataforma de Gêmeo Digital funcional, integrando Unity, Node.js, Python e MongoDB.
- **Resultados (Atividades):**
  - Replay de clima (INMET) sob-demanda foi implementado.
  - Detecção de Outlier (LOF) em tempo real foi validada.
  - Classificação (RandomForest) e Balanceamento (SMOTE) foram aplicados com sucesso.
- **Trabalhos Futuros:**
  - Usar a predição (de clima e poluição) para *alterar* a simulação (ex: adicionar névoa de poluição, mudar a rota dos carros).
  - Criar um painel de visualização de dados (dashboard) em HTML que gere gráficos automaticamente a partir dos resultados.
  - Integrar o Gemini para gerar explicações automáticas desses gráficos.

# Bibliografia (a.7)


CRUZ, A. B., et al. (2017). "Detecção de Anomalias no Transporte Rodoviário Urbano". *Anais do Simpósio Brasileiro de Banco de Dados (SBBD)*.

DELLA-JUSTINA, H. M. (2023). "Avaliação de Técnicas de Classificação Para Dados Desbalanceados". *Acervo Digital UFPR*.

VIANNA JR, A. S., et al. (2025). "Analisando Dados de Qualidade do Ar por Machine Learning". *Periodicos FURG*.

Unity Technologies. (2025). "Construa Gêmeos Digitais Industriais".

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python".

Lemaître, G. et al. (2017). "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets".