

PUMLE: A Framework for Data-Driven Applications in Underground Carbon Sequestration Ventures

Luiz Fernando Costa dos Santos



CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa
2025

Luiz Fernando Costa dos Santos

PUMLE: A Framework for Data-Driven Applications in Underground Carbon Sequestration Ventures

Monograph presented to Centro de Informática,
Universidade Federal da Paraíba, in fulfillment of
the requirements for the Bachelor's degree in Data
Science and Artificial Intelligence.

Advisor: Gustavo Charles Peixoto de Oliveira

João Pessoa

2025

Catálogo na publicação
Seção de Catalogação e Classificação

S237p Santos, Luiz Fernando Costa Dos.

PUMLE: a framework for Data-Driven applications in underground carbon sequestration ventures / Luiz Fernando Costa Dos Santos. - João Pessoa, 2025.

51 f. : il.

Orientação: Gustavo Charles Peixoto de Oliveira Oliveira.

TCC (Graduação) - UFPB/CI.

1. Sequestro de Carbono. 2. Engenharia de Dados. I. Oliveira, Gustavo Charles Peixoto de Oliveira. II. Título.

UFPB/CI

CDU 004.6



CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Monograph of Data Science and Artificial Intelligence entitled ***PUMLE: A Framework for Data-Driven Applications in Underground Carbon Sequestration Ventures***, authored by Luiz Fernando Costa dos Santos, and approved by the following Examination Jury members:

Prof. Dr. Gustavo Charles Peixoto de Oliveira
Universidade Federal da Paraíba

Prof. Dr. Yuri de Almeida Malheiros Barbosa
Universidade Federal da Paraíba

Prof. Dr. Adriano Maurício de Almeida Côrtes
Universidade Federal do Rio de Janeiro

Course Coordinator
Prof. Dr. Teobaldo Bulhões Junior

João Pessoa, May 13th, 2025

RESUMO

O PUMLE é um *framework* projetado para gerenciar de forma integrada grandes volumes de dados gerados a partir de simulações numéricas de injeção de CO₂ em reservatórios subterrâneos. Utilizando uma arquitetura inspirada no modelo *Medallion*, o fluxo de trabalho abrange as principais etapas do ciclo de vida dos dados, desde a geração por meio de simulações em lote até o armazenamento final dos resultados em formatos de arquivo eficientes, priorizando o mínimo esforço computacional. O PUMLE destina-se a gerar dados de alta qualidade para subsidiar experimentos de aprendizado de máquina voltados à previsão da dinâmica de plumas de CO₂ em futuros projetos brasileiros de sequestro de carbono, particularmente aqueles baseados em técnicas conscientes da fenomenologia física por trás do processo de injeção. As métricas de desempenho indicam que, embora a solução numérica seja responsável pela maior parte do tempo de processamento, as etapas subsequentes de ingestão, transformação e armazenamento de dados do PUMLE demonstraram alta eficiência, consumindo coletivamente apenas cerca de 1,4% do tempo total de processamento (*wall-clock time*) em testes de simulação de referência. Espera-se que esta ferramenta facilite a criação de conjuntos de dados consistentes, proporcionando escalabilidade e reprodutibilidade para treinamento, testes e prototipagem de soluções orientadas a dados, focadas nas etapas de desenvolvimento e monitoramento de empreendimentos de sequestro de carbono em domínios heterogêneos.

Palavras-chave: sequestro de carbono, simulação numérica, engenharia de dados, aprendizagem de máquina.

ABSTRACT

PUMLE is a *framework* designed to seamlessly manage large datasets generated from numerical simulations of CO₂ injection into underground reservoirs. Using an architecture inspired by the *Medallion* model, the workflow covers the key stages of the data lifecycle, from generation through batch simulations to the final storage of results in efficient file formats, prioritizing minimal computational effort. PUMLE is intended to generate high-quality data to support machine learning experiments aimed at predicting the dynamics of CO₂ plumes in future Brazilian carbon sequestration projects, particularly those based on techniques aware of the physical phenomenology behind the injection process. Performance metrics indicate that while the numerical solution accounts for the bulk of processing time, PUMLE’s subsequent data ingestion, transformation, and storage stages demonstrated high efficiency, collectively consuming only about 1.4% of the total wall-clock time in reference simulation tests. This tool is expected to facilitate the creation of consistent datasets, providing scalability and reproducibility for training, testing, and prototyping data-driven solutions focused on the development and monitoring stages of carbon sequestration ventures in heterogeneous domains.

Keywords: carbon sequestration, numerical simulation, data engineering, machine learning.

LIST OF FIGURES

1	CCS process chain	12
2	Medallion inspired architecture	22
3	PUMLE illustration pipeline	22
4	Bronze layer - Data Generation	24
5	Silver layer - Parsing JSON	25
6	Golden layer - Consolidation	25
7	CLI Menu	28
8	2D view of the Pressure field	30
9	3D view of the Saturation field.	31
10	AWS S3 Artifacts	35

LIST OF TABLES

1	Runtime table	29
2	Runtime table - Fast scenario	31
3	Runtime table - Slow scenario	32

Contents

1 INTRODUCTION	11
1.1 Scope	11
1.2 Motivation	11
1.3 Research Objectives	13
2 LITERATURE REVIEW	14
2.1 Carbon Capture, Utilization, and Storage	14
2.2 Geological Carbon Storage	14
2.3 Plume Migration Dynamics	15
2.3.1 Thermodynamics and Transport	15
2.3.2 Two-Phase Flow Equations	15
2.3.3 Trapping Mechanisms	16
2.4 Fluid Properties	16
2.5 Numerical Simulations	16
2.5.1 Importance of Heterogeneity and Coupled Processes	17
2.5.2 Surrogate and Reduced-Order Modeling	18
2.5.3 Applications to Site Management	18
2.6 Data Domain	18
2.6.1 Role of Large-Scale Datasets in Carbon Dioxide Plume Modeling	18
2.6.2 Challenges of Data Availability for Plume Prediction	19
2.6.3 Data Pipelines and Infrastructure	19
2.6.4 Synthetic Data Generation and Augmentation	20
3 METHODOLOGY	21
3.1 Project Design	21
3.2 Bronze Layer: Data Generation	22
3.3 Silver Layer: Data Parsing and Preparation	23
3.4 Golden Layer: Array Consolidation, Formatting, and Storage	24

4 RESULTS AND DISCUSSION	28
4.1 System Workflow and Execution	28
4.2 Performance with Single Simulation	29
4.3 Performance Analysis of Simulations	30
4.3.1 Fast Simulation Scenario	30
4.3.2 Slow Simulation Scenario	32
4.3.3 Comparative Discussion	32
4.4 Scaling to Multiple Scenarios with Parallelism	33
4.5 Data Volume and Efficiency	33
4.6 Infrastructure Considerations, Limitations, and Future Outlook	34
5 CONCLUSION	36
REFERENCES	37
APPENDICES	41
APPENDIX A - Example <code>setup.ini</code> Configuration File	42
APPENDIX B - Fluid Property Models in <code>co2lab3DPUMLE</code>	47
APPENDIX C - Code Structure	50
APPENDIX D - PUMLE Article Publication	51

1 INTRODUCTION

1.1 Scope

The emissions of greenhouse gases, particularly carbon dioxide (CO_2), have reached unprecedented levels, driving significant global climate changes with severe implications for natural systems. According to recent data from the Global CCS Institute, anthropogenic CO_2 emissions have continuously increased as a result of the industrial activity, energy generation from fossil fuels, and deforestation (GCCSI, 2024). These emissions induce climate changes and contribute to sea-level rise, ocean acidification, and extreme weather events, which collectively threaten biodiversity, water resources, food security, and human health.

The Intergovernmental Panel on Climate Change (IPCC) has reported that to mitigate the worst effects of climate change, global temperature rise must be limited to 1.5°C above pre-industrial levels. Achieving this target requires a significant and immediate reduction in greenhouse gas emissions, necessitating the deployment of a broad array of technological, economic, and social strategies. Among these strategies, Carbon Capture and Storage (CCS) has emerged as a technological ensemble capable of significantly reducing CO_2 emissions from industrial sources and fossil fuel power plants.

Carbon sequestration, in particular, involves capturing CO_2 at its emission sources, transporting it securely, and subsequently injecting it into deep geological formations for permanent storage (AMINU *et al.*, 2017; GCCSI, 2024). Figure 1, adapted from (CHOUDHARY, 2016), illustrates the main stages of the CCS chain. At industrial plants or facilities, the gas is separated from other substances for reuse (CCU), captured from biomass combustion (BECCS), or directly captured from the air (DACCS). Afterwards, CO_2 is either compressed to a liquid form and transported in trucks, ships, or in its natural form in ducts for disposal. When destined to permanent storage, the gas is usually injected into onshore or offshore underground rock formations to prevent it from returning back to the atmosphere. After sequestration, the storage location must be considered a safe “sealed vessel of CO_2 ” (DZIEJARSKI; KRZYŻYŃSKA; ANDERSSON, 2023).

1.2 Motivation

The effectiveness of CCS lies in its potential for large-scale emission reduction, essential for the so-called “hard-to-abate” industries, where complete decarbonization via renewable energy or electrification remains technically challenging or economically infeasible. Moreover, CCS can be synergistically employed for other purposes, such as enhanced oil recovery (EOR) and product feedstock. In this case, it unwraps into the

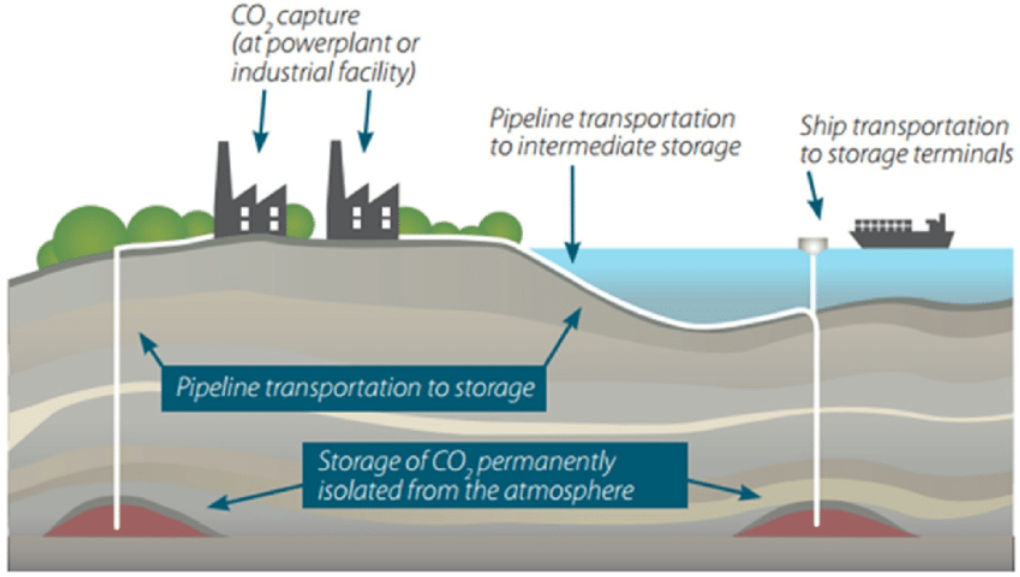


Figure 1: Main stages of the CCS process chain. Reproduced from (CHOUDHARY, 2016).

additional process branch termed “Utilization”, thus offering scalable alternatives under a full low-carbon CCUS-based economy.

Globally, several large-scale CCS projects have demonstrated the feasibility of capturing and securely storing millions of tons of CO₂ annually. Projects such as Sleipner in Norway (Norwegian Petroleum Directorate, 2000) and Quest in Canada have proven not only the technical viability but also the crucial role of CCS in comprehensive climate mitigation strategies. The Norwegian Northern Lights project (Northern Lights JV DA, 2024), for instance, was the first joint venture to run CCS at a commercial scale. Furthermore, Norwegian governmental policies have been fundamental in advancing these technologies. However, the global deployment of CCS is still insufficient relative to the scale required to meet climate targets. Challenges related to high costs, public acceptance, regulatory frameworks, and infrastructure needs continue to limit the broader adoption of CCS technologies.

Beyond these economic and regulatory barriers, scientific and technical challenges significantly impact CCS development. A central technical issue is managing the extensive datasets generated through detailed numerical simulations of CO₂ injection and plume migration within geological formations. Such simulations are indispensable for predicting plume behavior, assessing reservoir storage capacities, evaluating leakage risks, and complying with regulatory demands (HARP *et al.*, 2019; FAN; LU; LIU, 2023). However, traditional data analysis methods face limitations due to their large volume, high dimensionality, and complex interpretation.

To address these challenges, advanced data engineering approaches are necessary.

Innovative data management techniques capable of efficiently handling and interpreting large-scale, complex datasets are crucial to improving the scalability and efficiency of CCS projects. Physics-aware and physics-informed machine learning, particularly are novel and emerging tools that may become essential for harnessing these complex datasets, enhancing predictive accuracy and reducing the computational burden of traditional approaches.

1.3 Research Objectives

This monograph proposes an utility tool for carbon-removal engineering and practitioners involved with carbon sequestration. PUMLE, as a quibble for “plume”, is a data abstraction modeled to respond the future needs of the CCUS market in Brazil, by providing a mechanism for modern on-premise and cloud-based data serving for machine learning applications applicable to field monitoring. In particular, this approach envisions low-cost storage of massive unstructured datasets and seamless data integration.

This way, PUMLE has the main objective of generating high-quality data to support machine learning experiments aimed at tracking and predicting the dynamics of subsurface CO₂ plumes under varying conditions in anticipation to future carbon sequestration ventures over Brazil. The specific objectives are:

- to produce spatiotemporal data related to plume migration from numerical simulations of gas injection;
- to execute parallel batch simulations from mutiple physical parameters;
- to transform the simulation outputs into suitable array-based structured datasets;
- to store the datasets in a low-cost storage system; and
- to feed physics-based machine learning experiments with high-quality, consistent datasets.

2 LITERATURE REVIEW

2.1 Carbon Capture, Utilization, and Storage

Carbon Capture, Utilization, and Storage (CCUS) is a suite of technologies designed to (i) capture CO₂ from large industrial emission sources or directly from the atmosphere, (ii) utilize or process the captured CO₂ for various applications (e.g., enhanced oil recovery, synthetic fuels, chemicals, or other products), and (iii) store any remaining CO₂ safely in underground geological formations. The overarching goal is to prevent substantial amounts of CO₂ from entering the atmosphere, thereby mitigating global climate change (AMINU *et al.*, 2017; GCCSI, 2024). CCUS has gained traction because it can be deployed in “hard-to-abate” sectors, such as iron, steel or cement production, which still rely heavily on fossil fuels.

Key processes of the CCUS chain include:

1. **Capture:** separation of CO₂ from other gases using pre-combustion, post-combustion, or oxyfuel technologies.
2. **Compression and transport:** liquefaction of CO₂ and transportation by modals up to storage sites.
3. **Utilization:** destination of CO₂ for reuse in various industrial processes or feed-stocking.
4. **Storage:** injection of CO₂ into deep geological formations (e.g., saline aquifers, depleted oil and gas fields, unmineable coal seams, basalt formations) for temporary or permanent storage.

2.2 Geological Carbon Storage

Geological carbon storage (GCS), or carbon sequestration, is a major sub-component of CCUS technologies, focusing on long-term containment of CO₂ in deep geological formations. Among potential sinks, saline aquifers have broad geographic distribution and large storage capacity, while depleted oil/gas reservoirs can leverage existing infrastructure. Unmineable coal seams and basalt also present possible storage sites (BACHU, 2008; (IPCC), 2005).

A GCS project typically proceeds through three phases:

- **Injection phase:** CO₂ is injected under high pressure into a porous reservoir, displacing resident brine.

- **Post-injection phase:** the CO₂ plume continues to migrate under buoyancy and reservoir heterogeneity.
- **Long-term containment:** over time, CO₂ is immobilized by structural, residual, solubility, or mineral trapping mechanisms.

2.3 Plume Migration Dynamics

2.3.1 Thermodynamics and Transport

In many GCS operations, CO₂ is injected at depths where temperature and pressure exceed the CO₂ critical point ($\sim 31.0^\circ\text{C}$ and ~ 73.8 bar). Under these conditions, CO₂ becomes supercritical (or dense phase), achieving significantly higher density compared to gaseous CO₂. In porous media, CO₂ migration is governed by:

- **Buoyancy:** CO₂ has lower density than brine and tends to migrate upward until it meets a caprock (or seal).
- **Viscous forces:** pressure gradients drive fluid flow within the pore network.
- **Heterogeneity:** spatial variations in permeability and porosity create preferential pathways for fluid migration.

2.3.2 Two-Phase Flow Equations

For an isothermal two-phase system of CO₂ (subscript g) and brine (subscript w), the mass conservation equation for each phase α can be written as:

$$\frac{\partial(\phi \rho_\alpha S_\alpha)}{\partial t} + \nabla \cdot (\rho_\alpha \mathbf{u}_\alpha) = q_\alpha, \quad \alpha \in \{g, w\},$$

where ϕ is the porosity, ρ_α is the density, S_α is the saturation, and $\mathbf{u}_\alpha$ is the Darcy velocity. The source/sink term q_α represents injection or production. Under Darcy's law,

$$\mathbf{u}_\alpha = -\frac{k k_{r,\alpha}}{\mu_\alpha} (\nabla p_\alpha - \rho_\alpha \mathbf{g}),$$

where k is the absolute permeability, $k_{r,\alpha}$ is the relative permeability, μ_α is the dynamic viscosity, p_α is the fluid pressure, and $\mathbf{g}$ is the gravitational acceleration vector. Capillary pressure p_c relates phase pressures via

$$p_g = p_w + p_c(S_w).$$

The numerical simulator used in the PUMLE framework, namely the `TwoPhaseWaterGasModel` from the MRST `co2lab` module (LIE *et al.*, 2016), employs these Two-Phase Flow Equations to generate the simulation results.

2.3.3 Trapping Mechanisms

Long-term containment mechanisms include:

- *Structural (or stratigraphic) trapping*: caprocks physically block the buoyant CO₂.
- *Residual trapping*: CO₂ is immobilized in pore spaces by capillary forces.
- *Solubility trapping*: CO₂ dissolves in the brine and can induce density-driven convection.
- *Mineral trapping*: over geologic timescales, dissolved CO₂ can react with minerals to form carbonates.

2.4 Fluid Properties

Under supercritical conditions, CO₂ density and viscosity strongly depend on temperature and pressure. Equation-of-state-based formulations from (SPAN; WAGNER, 1996) can predict ρ_{CO_2} and μ_{CO_2} for ranges of p and T . Brine properties vary with temperature, pressure, and salinity. Empirical correlations to experimental data are common. For density, we can cite (ROWE; CHOU, 1970), (DRIESNER, 2007), and (MAO; DUAN, 2009). For viscosity, (KESTIN; KHALIFA; CORREIA, 1981), and (MAO; DUAN, 2009).

Representative formulas can be of the form:

$$\rho_w(p, T, X_{\text{NaCl}}) = \rho_0(T) \cdot [\text{salinity corrections}] \cdot [\text{pressure corrections}],$$

$$\mu_w(p, T, X_{\text{NaCl}}) \approx \mu_0(T) \times [\text{salinity corrections}] \times [\text{pressure corrections}].$$

For this research, the fluid property models used are discussed in Appendix B.

2.5 Numerical Simulations

In the context of CO₂ storage, numerical simulators are extensively used to model fluid flow, geomechanics, and reactive transport. They solve the governing mass and energy balance equations for multiphase flow in porous and fractured media. Key tools include simulators such as TOUGH2, CMG-GEM, and ECLIPSE, each supporting different

physical and chemical processes (OLDENBURG; SPYCHER; PRUESS, 2016; FAN *et al.*, 2019; HOSSEINI *et al.*, 2013).

Reservoir simulations generally:

- Employ a spatial discretization, e.g., finite-difference, finite-element, or finite-volume, to solve coupled partial differential equations describing flow and transport.
- Use time discretization (implicit or explicit) to handle the temporal progression of injection and post-injection.
- Integrate complex physics (capillarity, dissolution, adsorption, geomechanics, etc.) depending on project needs.

The domain is typically discretized into a 3D grid (Cartesian or radial) with known or stochastically generated porosity/permeability fields. Initial and boundary conditions:

- **Initial:** $p(0) = p_i$, $S_{\text{CO}_2}(0) = 0$, etc.
- **Injection:** specified injection rate $q_{\text{inj}}(t)$ at the injection well.
- **Boundary:** no-flow (closed system) or constant pressure (open system) boundary conditions.

Coupled with boundary and initial conditions, the basilar equations (cf. 2.3.2) form a large system of equations typically solved iteratively through Newton or quasi-Newton schemes (ZHANG; WU; DING, 2015; WHITE *et al.*, 2014).

2.5.1 Importance of Heterogeneity and Coupled Processes

Reservoir heterogeneity strongly influences CO₂ plume migration and pressure evolution. Fine-scale permeability contrasts can cause fingering or channeling of CO₂ (KUMAR; BRYANT; LAKE, 2010), while topographic features direct plume movement (OBMANUEL; HU; BATTIATO, 2017). In many scenarios, geomechanical or chemical effects (dissolution, mineralization) may need incorporation:

- **Geomechanical coupling:** High-pressure injection can alter stresses, leading to fracture initiation or reactivation.
- **Geochemical coupling:** Dissolved CO₂ may promote mineral precipitation or dissolution that modifies rock porosity/permeability.

Advanced simulators can manage such coupling at scale but at significant computational cost (NGHIEM *et al.*, 2009).

2.5.2 Surrogate and Reduced-Order Modeling

Because of the heavy computational burden of full-physics simulation, recent trends include:

- **Surrogate models:** Data-driven or physics-informed machine learning approximates simulator outputs, accelerating scenario evaluations (SHOKOUI *et al.*, 2021).
- **Upscaling and reduced-order methods:** Coarse-graining or polynomial chaos expansions, reducing high-dimensional parameter spaces (ZHANG; GAO; DURLOFF-SKY, 2016).

These methods support real-time decision-making and uncertainty analyses while retaining key physical accuracy.

2.5.3 Applications to Site Management

Throughout a storage project’s lifecycle, numerical simulation underpins design and operation:

- **Pre-injection feasibility:** Predict injectivity, identify optimal well placement, estimate plume footprint and storage capacity.
- **During injection:** Track pressure buildup, manage injection rates, mitigate near-well fracturing risks.
- **Post-injection period:** Forecast CO₂ migration, assess potential leakage pathways, inform monitoring programs.

In sum, numerical methods are a cornerstone of modern carbon storage, enabling the evaluation of complex physical, chemical, and operational factors that influence safe and efficient CO₂ sequestration.

2.6 Data Domain

2.6.1 Role of Large-Scale Datasets in Carbon Dioxide Plume Modeling

With the growing use of machine learning (ML) methods in CO₂ plume prediction, the need for *high-quality, large-scale datasets* is more pressing than ever. Subtle interactions among reservoir heterogeneity, fluid properties, and injection operations demand diverse, comprehensive data to reliably train models (SHOKOUI *et al.*, 2021; TANG *et al.*,

[2021]). Large, well-curated datasets help mitigate overfitting, support robust uncertainty estimation, and ultimately yield more generalizable predictions.

The process of amassing such datasets, however, can be challenging. Many advanced data engineering principles—often derived from broader database and analytics work—are required to organize, clean, and version these data in ways conducive to ML pipelines ([KIMBALL, 1996; KLEPPMANN, 2017]).

2.6.2 Challenges of Data Availability for Plume Prediction

Despite the recognized importance of big data in ML, the CO₂ storage context poses hurdles:

- **Scarce field-scale injection data:** Relatively few commercial-scale CO₂ injection operations have been realized, leading to sparse or short-term measurements ([HARP *et al.*, 2019]).
- **Limited well network instrumentation:** Even in existing projects, sensors may be few. This often yields partial observations of pressure or saturation fields.
- **Confidential or siloed datasets:** Corporate or governmental pilot projects frequently treat operational data as proprietary, hindering open data exchange ([PAWAR *et al.*, 2019]).
- **Heterogeneous formats:** Legacy data could span simulator-specific binary outputs, CSV logs, or unstructured text, complicating uniform ingestion ([AMPOMAH *et al.*, 2017]).

Such constraints underscore the shortfall in the *volume* and *variety* of data critical to comprehensive data-driven models.

2.6.3 Data Pipelines and Infrastructure

Robust data pipelines address these limitations by structuring the end-to-end handling of simulation outputs or field measurements. In line with standard frameworks from the data warehousing community ([KIMBALL, 1996]), a typical pipeline for plume prediction may involve:

1. **Data ingestion:** Aggregating multiple file types—HDF5, CSV, or netCDF—into a consistent repository or data lake ([KLEPPMANN, 2017]).

2. **Preprocessing:** Detecting and removing outliers, adjusting coordinate systems, harmonizing metadata, and handling missing or corrupted entries. Harmonized metadata and adjusting coordinate systems means deal with the sparse and different sizes and structures from the Simulator output.
3. **Feature engineering:** Mapping raw saturation fields to 2D/3D arrays for image-based ML or extracting summary metrics (like average saturations, pressure gradients) for tabular approaches. Image-based ML in this context stands for trying a CNN approach for predicting the gas behavior through time with pictures for each time step.
4. **Storage and versioning:** Storing large simulation ensembles on HPC cluster file systems or cloud-based object stores, with version control (e.g., Git, DVC) for reproducibility (PRATX *et al.*, 2018).

Such systematic design ensures that downstream ML tasks can proceed efficiently, with minimal data wrangling overhead.

2.6.4 Synthetic Data Generation and Augmentation

Because large real-world datasets remain limited, synthetic data produced from reservoir simulators—such as CMG-GEM, TOUGH2, OPM-Flow, and Jutul Darcy (the latter two being open-source)—is commonly used to expand training sets (ZHONG *et al.*, 2019; FAN *et al.*, 2022). Main approaches include:

- **Stochastic reservoir modeling:** Assign random but geologically consistent porosity-permeability fields to emulate natural heterogeneity.
- **Parameter scanning:** Vary injection rates, boundary conditions, and fluid properties to capture a broad scenario space.
- **Augmentation techniques:** Add domain-specific noise or apply transformations to boost data diversity, aligning with best practices from advanced data engineering references (KLEPPMANN, 2017).

3 METHODOLOGY

3.1 Project Design

The PUMLE framework was designed as a command-line interface (CLI) driven system to orchestrate the generation, processing, and storage of datasets from CO₂ injection simulations. Its primary aim is to produce analysis-ready datasets representing key physical properties (e.g., gas saturation, pressure) over space and time, suitable for subsequent machine learning applications focused on understanding subsurface plume dynamics.

The data workflow is inspired by the layered Medallion architecture commonly used in data warehousing (KIMBALL, 1996), adapted for simulation data management (Figure 2). This architecture promotes progressive data refinement and modularity:

- **Bronze Layer:** contains the raw, unprocessed output directly from the numerical simulator (MRST `co2-lab`), generated via orchestrated parallel runs. In this implementation, the primary Bronze artifact is a set of JSON files produced by the `co2lab3DPUMLE.m` script for each simulation run.
- **Silver Layer:** this intermediate processing stage focuses on reading the raw Bronze JSON files, parsing the complex structures, validating data integrity (e.g., grid indices), and preparing structured data representations *in memory* (e.g., Python lists, dictionaries containing state variables for active cells) ready for final consolidation. Unlike some Medallion implementations, PUMLE, in its current primary workflow, does not persist a separate, structured Silver file format; the refinement occurs during the transition to the Golden layer processing.
- **Golden Layer:** this final layer takes the prepared data structures from the Silver stage, consolidates them into unified, multi-dimensional NumPy arrays (`.npy`) or Zarr arrays (`.zarr`) representing the spatiotemporal evolution of variables across the entire simulation domain, and handles persistent storage either locally or in cloud object storage (AWS S3). These Golden arrays are the primary, analysis-ready outputs intended for ML applications.

This layered approach allows for flexibility in handling different simulation outputs and isolates the computationally intensive simulation from data processing and storage tasks. The framework components, including parameter handling, simulation execution, data parsing, array generation, and storage, are implemented as Python modules orchestrated by the main PUMLE application. The overall end-to-end data flow is illustrated in Figure 3.

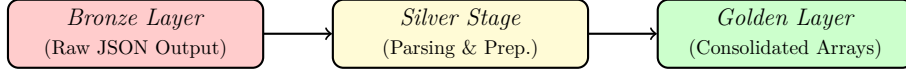


Figure 2: PUMLE’s data flow architecture, inspired by the Medallion model, showing the implemented stages.

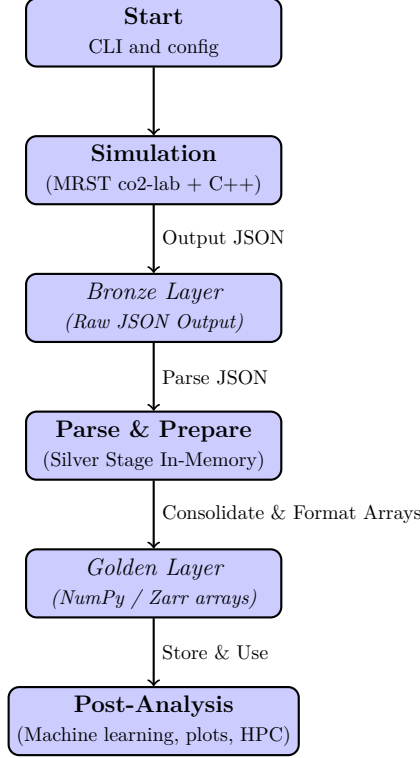


Figure 3: Illustration of the PUMLE pipeline used for scenario runs and data generation.

3.2 Bronze Layer: Data Generation

The foundation of the PUMLE framework lies in generating synthetic datasets through numerical simulation of CO₂ injection. This process constitutes the Bronze Layer.

The Reservoir Model used for experiments was the UNISIM-I-D benchmark model (CEPETRO/UNICAMP). This model represents a heterogeneous sandstone reservoir analogous to formations in the Campos Basin, making it relevant for studying complexities pertinent to potential Brazilian CCS contexts and providing a standard test case for data management challenges.

Regarding the Simulation Engine, the core multiphase flow simulations were performed using the MRST co2lab (LIE *et al.*, 2016) module’s TwoPhaseWaterGasModel model within a customized MATLAB/Octave script (co2lab3DPUMLE.m). This setup models isothermal two-phase flow of CO₂ and brine. While sufficient for generating baseline plume migration data, this simplification neglects potentially important non-isothermal effects, geomechanical coupling (NGHIEM *et al.*, 2009; ZHANG; WU; DING, 2015), and reactive transport processes present in real GCS scenarios. The impact of these omissions

on the direct applicability of the generated data for certain high-fidelity ML tasks should be considered.

Parameterization of the simulations is defined in a central `setup.ini` configuration file (see Appendix A), parsed using PUMLE’s `ini.py` module. Key parameters include fluid properties (reference pressure/temperature, residual saturations, capillary pressure parameters like p_e), rock compressibility, initial/boundary conditions, well locations/rates, and simulation schedule. The specific values used (e.g., $P_{ref} = 35$ MPa, $T_{ref} = 95.15$ K) were chosen as representative baseline conditions for the UNISIM-I-D model depth and typical GCS scenarios, although a detailed sensitivity analysis or justification based on specific Brazilian site characterization was beyond the scope of this work. Parameter validation against defined physical limits is handled internally by the `parameters.py` module.

The framework supports automated Parameter Variation via the `ParametersVariation` class, generating combinations for sensitivity studies or ML dataset generation. For each unique parameter set, corresponding simulation input files (structured `.mat` files expected by `co2lab3DPUMLE.m`) are created by `mat_files.py`. Caching based on parameter hashes prevents redundant runs.

To accelerate scenario generation, PUMLE utilizes Parallel Execution managed by `simulation_script.sh`. A C++ wrapper (`simulation.cpp`), compiled with OpenMP support (CHAPMAN; LAGRONE, 2011), orchestrates the concurrent execution of multiple `co2lab3DPUMLE.m` instances via Octave (EATON *et al.*, 2023).

The Bronze Output Format for each simulation run consists of a set of JSON files containing time-series data (pressure, saturation ‘s’) for active cells, grid dimensions, and active cell masks. JSON was chosen as the initial output format primarily due to the ease of generation from the existing MRST script using built-in functions (`jsonencode`), prioritizing implementation simplicity and interoperability over the storage efficiency of binary formats at this stage. However, the raw, potentially nested structure requires parsing before efficient numerical use. These raw JSON files constitute the persisted Bronze Layer data.

Figure 4 provides a flowchart of this Bronze Layer data generation process.

3.3 Silver Layer: Data Parsing and Preparation

Following the Bronze Layer generation, the raw simulation outputs require processing to become suitable for analysis and final storage. This corresponds to the Silver Stage in PUMLE’s workflow, focusing on parsing the Bronze JSON files and preparing structured data representations in memory. This stage acts as a crucial *transient processing step*, decoupling the raw output format from the final array generation logic.

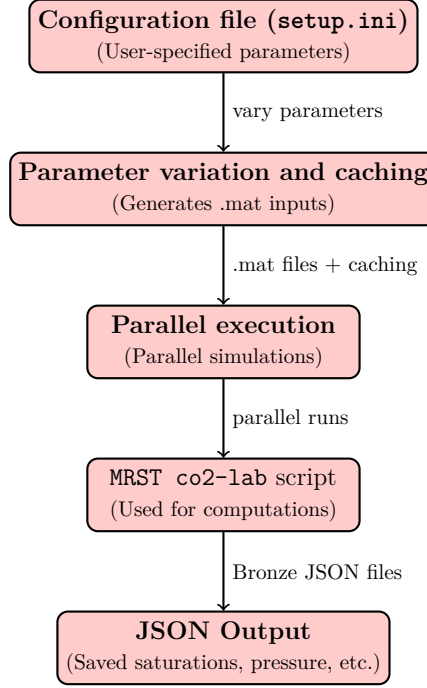


Figure 4: Data generation overview flowchart (Bronze Layer).

The primary component responsible for this stage is the `SimResultsParser` class (`sim_results_parser.py`). Its main tasks within the PUMLE workflow include:

1. **Reading Bronze JSON Files:** Locates and reads the necessary JSON output files (state data, grid dimensions, active cells) for a given simulation run, identified by its hash.
2. **Extracting Key Variables:** Iterates through time steps, extracting primary variables (pressure, saturation array 's').
3. **Validating Indices and Dimensions:** Retrieves grid dimensions (nx, ny, nz) and active cell indices, performing consistency checks between data array sizes and active cell counts.
4. **Structuring In-Memory Data:** Organizes the extracted, validated data for each time step into Python lists of dictionaries, ready for input into the Golden Layer.

This parsing and preparation stage handles necessary cleaning and validation before the final consolidation and storage, enhancing modularity. Figure 5 illustrates this process.

3.4 Golden Layer: Array Consolidation, Formatting, and Storage

The Golden Layer represents the culmination of the PUMLE data pipeline, responsible for transforming the prepared in-memory data structures from the Silver stage into

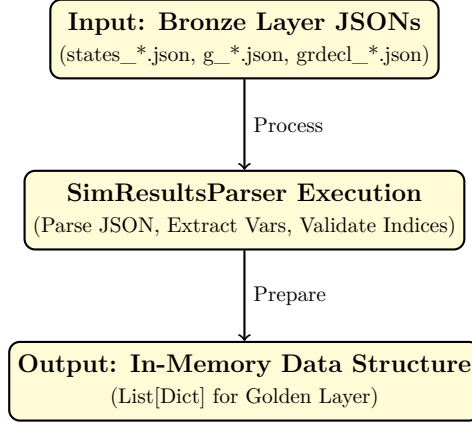


Figure 5: Silver Stage process: Parsing Bronze JSON and preparing in-memory data structures.

final, analysis-ready numerical arrays and managing their persistent storage. This layer produces the primary assets intended for ML modeling and other data-driven analyses. The workflow, implemented mainly within the `Arrays` class (`arrays.py`), is shown in Figure 6 and involves:

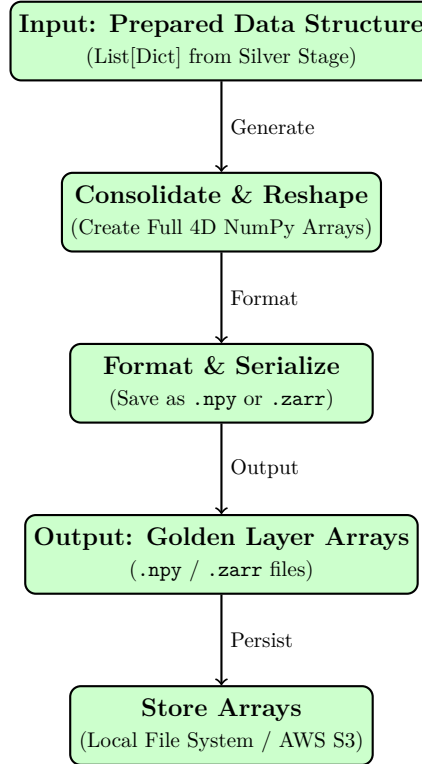


Figure 6: Golden Layer process: Consolidating parsed data into NumPy/Zarr arrays and storing them.

1. **Consolidating Data Across Time:** The `consolidate_all_data` method takes the list of per-timestep data structures. It initializes full-domain NumPy arrays (shape `'(nx*ny*nz, num_timesteps)'`) for pressure, Sw, and Sg, then iterates through

time steps, mapping active cell data into these arrays. *Critical Note:* This consolidation requires memory proportional to the full grid size times the number of time steps for potentially multiple variables, posing a scalability bottleneck for large simulations (discussed further in Section [4.6](#)).

2. **Reshaping to Spatiotemporal Arrays (4D):** Consolidated arrays are reshaped to '(nx, ny, nz, ntimesteps)' using Fortran ordering ('F'), matching typical reservoir simulation indexing.
3. **Array Formatting and Serialization:** The 4D NumPy arrays are serialized to disk. PUMLE supports two primary formats via the `Arrays` class:
 - **.npz:** Saved using `np.savez()`. Chosen as the default in demonstrated results for its simplicity and direct integration with the Python scientific computing stack. However, it lacks built-in chunking, compression, and metadata embedding.
 - **.zarr:** Saved using the `zarr` library via the `save_zarr` method. This format supports chunking (enabling partial reads/writes), various compression algorithms, and better scalability for cloud or parallel filesystem environments, making it a more suitable choice for very large datasets.

The choice between `.npz` and `.zarr` can be configured, allowing users to balance simplicity with performance and scalability needs.

4. **Persistent Storage (Local or Cloud):** PUMLE stores the generated array files (`.npz` or `.zarr`) either locally in the 'golden_data' directory or, if configured, uploads them to AWS S3 using the `CloudStorage` class and 'boto3'. Basic error handling for cloud uploads is included.
5. **Organization and Traceability:** Traceability relies on a combination of:
 - **Database Tracking:** An SQLite database (`db.py`) links a unique 'sim_hash' to 'sim_id', status, and fluid parameters (as a string).
 - **File Naming Convention:** Output filenames incorporate a parameter string ('utils.py') or the 'sim_hash'.
 - **Directory Structure:** Files are organized locally or in S3 using prefixes/folders reflecting these identifiers.

This mechanism is functional but lacks robustness for complex experiments compared to systems with explicit data versioning or rich metadata embedding (potentially feasible with Zarr). The `metadata.py` module currently serves internal validation/export roles, not primary metadata persistence for the main array workflow.

6. Security Considerations (Cloud): Standard AWS security practices (IAM, encryption, logging) are required for S3 usage.

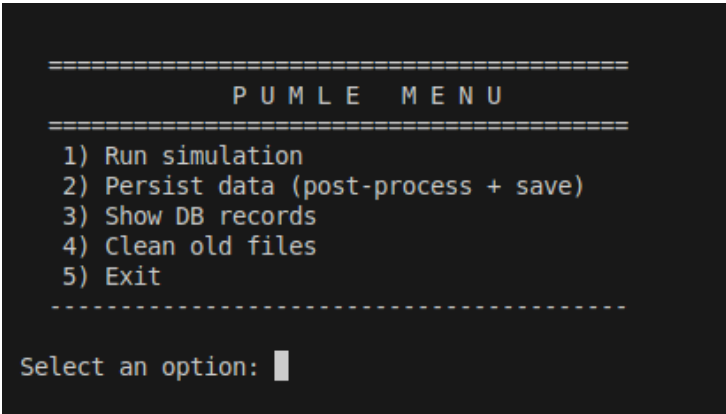
The Golden Layer thus produces the final `.npy` or `.zarr` arrays, making the simulation results accessible for data-driven GCS studies, with format options offering trade-offs between simplicity and advanced features.

4 RESULTS AND DISCUSSION

To demonstrate the feasibility and efficiency of the proposed PUMLE framework, we first detail the system’s execution workflow as implemented, followed by results from numerical experiments using the UNISIM-I-D benchmark model (CEPETRO/UNICAMP). This section presents (i) the operational flow of the PUMLE system, (ii) performance metrics for a single simulation run, (iii) observed performance under parallel execution for multiple scenarios, and (iv) a discussion on data management aspects, infrastructure considerations, and potential limitations based on the current implementation.

4.1 System Workflow and Execution

The PUMLE system is designed to streamline the simulation lifecycle from configuration to final data storage. It is initiated by executing the script `main.py`, which acts as the entry point for the PUMLE command-line interface (CLI). This interface allows users to configure simulation parameters (via `setup.ini`), define parameter variations for batch runs, launch simulations, and monitor their progress (Figure 7).



```
=====
                        P U M L E   M E N U
=====
1) Run simulation
2) Persist data (post-process + save)
3) Show DB records
4) Clean old files
5) Exit
-----
Select an option: █
```

Figure 7: Command Line Interface PUMLE’s menu.

Once executed via the CLI, the system follows the end-to-end architecture previously outlined (Section 3.1). The pipeline begins with the orchestration of reservoir simulations, managed by `simulation_script.sh`. This script compiles a C++ wrapper (`simulation.cpp`) using `g++` with OpenMP support (CHAPMAN; LAGRONE, 2011), enabling the parallel execution of multiple simulator instances.

Each parallel job then invokes the MATLAB/Octave-based script `co2lab3DPUMLE.m`. This script performs the core numerical simulation using MRST (LIE *et al.*, 2016), consuming `.mat` parameter files (generated by `mat_files.py`) and outputting the raw simulation results as a set of JSON files (Bronze Layer artifacts).

Following the simulation stage, the Python-based post-processing pipeline handles the data refinement and storage:

- **Silver Stage (Parsing/Preparation):** The raw Bronze JSON results are read and processed by `sim_results_parser.py`. This module extracts key variables, validates indices, and prepares structured data representations (Python lists/dictionaries) in memory.
- **Golden Layer (Array Generation/Storage):** The `arrays.py` module takes the in-memory data, consolidates it into 4D NumPy arrays, and serializes these arrays to either `.npy` or `.zarr` files, based on configuration. The `cloud_storage.py` module handles optional upload to AWS S3.
- **Metadata/Traceability:** Simulation runs are tracked using an SQLite database (`db.py`), storing a simulation hash, ID, status, and fluid parameters. Output array filenames also incorporate parameter information via `utils.py` functions. The `metadata.py` module exists primarily for internal parameter validation or potential separate tabular export functionality, not for creating persistent metadata files alongside the primary array outputs in the main workflow.

This modular design facilitates high-throughput experimentation, but the specific implementation choices (initial JSON output, in-memory consolidation, reliance on DB/naming for traceability) have implications discussed below.

4.2 Performance with Single Simulation

With the workflow established, we evaluated its performance. The initial reference simulation –using parameters representative of the UNISIM-I-D model depth (e.g., 35 MPa pressure, 95.15 K temperature, 10 % salinity) and a $1.5 \times 10^5 \text{ m}^3/\text{year}$ CO₂ injection rate for 15 years – yielded about 3.3 million active data points per time step across three primary features (gas saturation, brine saturation, pressure). The raw simulator output (Bronze JSON files) amounted to 22.5 MB. Executing the entire workflow for this single run (outputting to `.npy`) required approximately 16 minutes on a standard laptop (16 GB RAM, 13th Gen Intel®i7-13620H). Table 1 details the runtime breakdown, mapping stages to the conceptual layers and implemented steps.

Table 1: Runtime breakdown for the single-run scenario (outputting `.npy`), aligning process stages with the framework’s layers and reflecting the implemented workflow.

Process Stage	Duration (min:sec)	Relative Share
Simulation (Bronze JSON Output Gen.)	16:00	98.6%
Data Proc. (Silver Parsing + Golden Array Gen.)	00:10	1.0%
Array Storage (Golden Storage to Disk - <code>.npy</code>)	00:06	0.4%
Total	16:16	100%

As expected, the Bronze layer operations—specifically the direct simulator computations via `MRST co2-lab` and the subsequent writing of initial JSON files—dominate the runtime at nearly 16 minutes. This reflects the inherent computational cost of solving the multiphase flow equations. By contrast, the subsequent data processing steps encompassed by the PUMLE framework—parsing the JSON (Silver stage), consolidating and generating NumPy arrays (Golden processing), and writing the final `.npz` arrays to local disk (Golden storage)—collectively consumed only 16 seconds, representing approximately 1.4% of the total wall-clock time. This demonstrates that the data handling pipeline implemented in PUMLE adds minimal computational overhead relative to the core numerical simulation itself for this scale of problem.

Examples of the final visualized outputs (pressure and gas saturation) derived from the generated Golden Layer arrays are shown in Figure 8 and Figure 9.

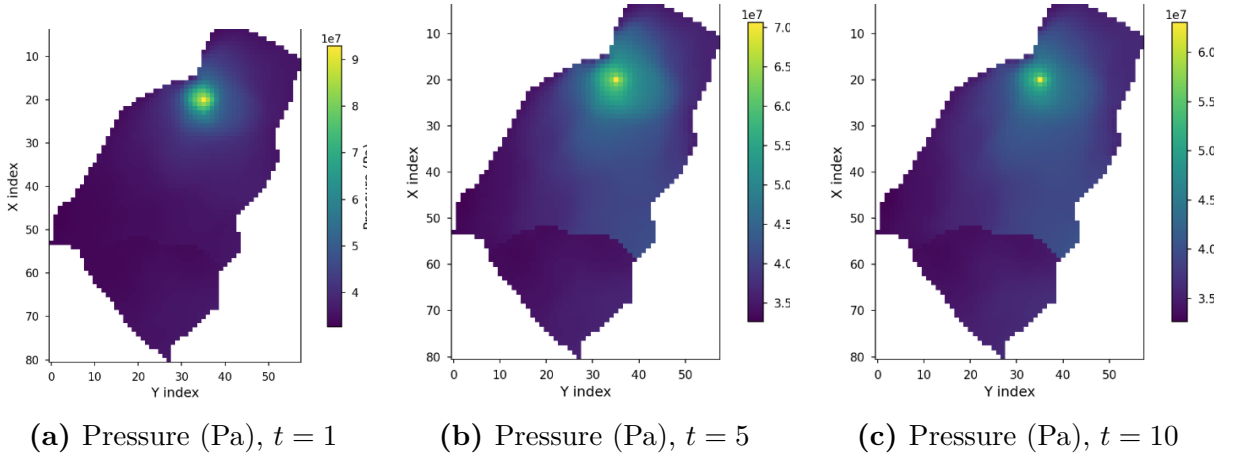


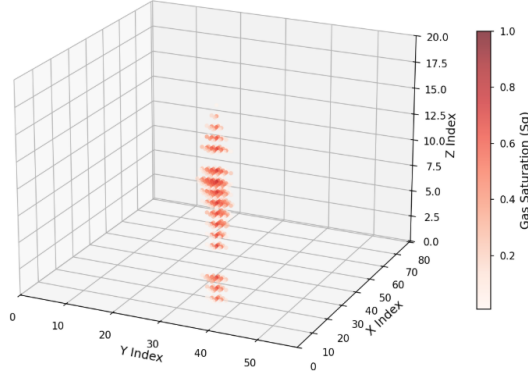
Figure 8: Visualization of CO₂ pressure field (Layer 1) at different time steps, generated from PUMLE output.

4.3 Performance Analysis of Simulations

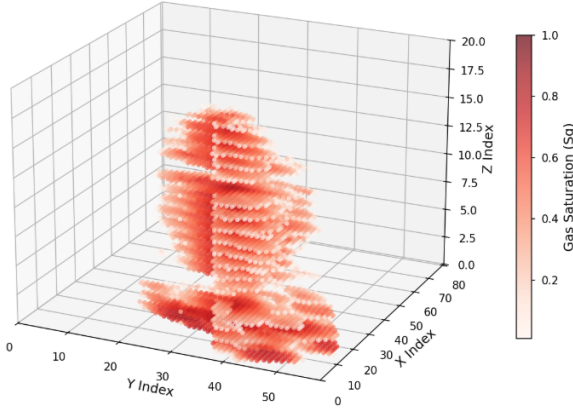
To further investigate PUMLE’s performance characteristics under different computational loads, two additional simulation scenarios were executed: a "Fast" scenario with significantly reduced injection and migration times, and a "Slow (Long Duration)" scenario representing a more computationally intensive simulation. These are compared against the "Mid-Range" reference scenario detailed in Section 4.2 (see Table 1). For all scenarios, the UNISIM-I-D model was used on the same standard laptop hardware.

4.3.1 Fast Simulation Scenario

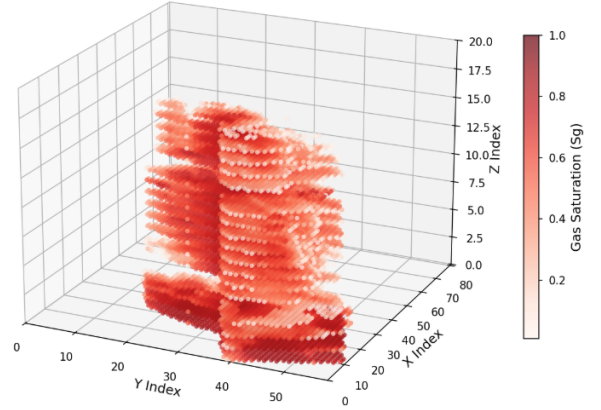
The "Fast" scenario was configured with an `injection_time` of 2 years and a `migration_time` of 3 years, with `migration_timesteps` set to 5, resulting in a total of 10



(a) Saturation Gas, $t = 1$



(b) Saturation Gas, $t = 5$



(c) Saturation Gas, $t = 10$

Figure 9: Visualization of 3D CO₂ gas saturation (Sg) plume at different time steps, generated from PUMLE output.

output states. This setup represents a quick, preliminary simulation run using the PUMLE framework. The runtime breakdown, based on an actual execution of this scenario, is presented in Table 2.

Table 2: Runtime breakdown for the "Fast" simulation scenario (2 years injection, 3 years migration, 10 total output states), based on actual execution.

Process Stage	Duration (min:sec)	Relative Share
Simulation (Bronze JSON Output Gen.)	03:11	99.6%
Data Proc. (Silver Parsing + Golden Array Gen.)	00:01	0.3%
Array Storage (Golden Storage to Disk .npy)	00:00	0.1%
Total	03:12	100%

For this "Fast" scenario, the numerical simulation (191.38 seconds) consumed virtually all of the processing time (99.6%). The PUMLE data handling stages (processing and storage) were exceptionally efficient, collectively taking only 0.85 seconds. This

represents a mere 0.44% of the total workflow time, underscoring PUMLE’s minimal overhead even for rapid simulation cycles.

4.3.2 Slow Simulation Scenario

The "Slow (Long Duration)" scenario was designed to test PUMLE with a significantly larger computational task and data volume. It was configured with an `injection_time` of 40 years and a `migration_time` of 60 years, with `migration_timesteps` still set to 5, resulting in a total of 10 output states. Table 3 details its performance.

Table 3: Runtime breakdown for the "Slow (Long Duration)" simulation scenario (40 years injection, 60 years migration, 10 total output states), based on actual execution.

Process Stage	Duration (min:sec)	Relative Share
Simulation (Bronze JSON Output Gen.)	62:31	98.6%
Data Proc. (Silver Parsing + Golden Array Gen.)	00:35	0.9%
Array Storage (Golden Storage to Disk .npy)	00:17	0.5%
Total	63:23	100%

In this "Slow (Long Duration)" scenario, the simulation phase took 63:23 minutes. PUMLE’s data handling, while processing a much larger dataset, completed in 52 seconds. Notably, the relative share of PUMLE’s operations in the total workflow time kept approximately 1.4%.

4.3.3 Comparative Discussion

Comparing the "Fast" (Table 2), "Mid-Range" (Table 1), and "Slow (Long Duration)" (Table 3) scenarios highlights a key characteristic of the PUMLE framework. The "Fast" scenario, with actual measured data, showed an extremely low PUMLE overhead of approximately 0.44%, with data handling completing in just 0.85 seconds. For the "Mid-Range" scenario, PUMLE’s estimated overhead was approximately 1.4%. For the "Slow (Long Duration)" scenario, which involved significantly more data and a much longer simulation phase, the estimated PUMLE overhead keeps in 1.4%.

This trend, particularly supported by the actual data from the "Fast" scenario, indicates that PUMLE’s data ingestion, processing, and storage tasks are highly optimized. While the absolute time for these tasks naturally increases with larger data volumes, its proportion of the *total* workflow time remains very small and can even decrease for very long simulations. This underscores PUMLE’s efficiency, demonstrating that the framework adds minimal relative overhead, particularly for the large-scale simulations typical in comprehensive CCS studies. The primary bottleneck remains, as expected, the numerical simulation itself.

4.4 Scaling to Multiple Scenarios with Parallelism

To assess the framework’s ability to handle multiple runs, typically required for sensitivity analysis or ML dataset generation, we varied the reference pressure (P_{ref}) from 24.5 MPa to 45.5 MPa, generating three additional simulation runs beyond the reference case.

- *Parallel Execution:* Leveraging the OpenMP-enabled C++ wrapper, all three new simulations executed concurrently on the multi-core laptop. The total wall-clock time for these three runs was approximately 26 minutes. This generated an additional 13.2 million data points per time step, resulting in 90 MB of combined final `.npy` data. The total runtime represented a roughly 1.56x increase over the single-run scenario baseline (16 minutes).
- *Serial Hypothetical Comparison:* Executing these three additional simulations sequentially would have required approximately $3 \times 16 \text{ min} \approx 48 \text{ minutes}$ for the simulation stage alone. The parallel execution thus provides a significant speedup (estimated speedup factor $\approx (16 + 48)/(16 + 26) \approx 1.5x$), primarily limited by the number of available CPU cores.

Critically, the data processing stages (Silver parsing + Golden array generation/storage) exhibited much more modest scaling, increasing by only about 1.18x despite the quadrupled data volume being processed post-simulation. This confirms that the overhead associated with PUMLE’s Python-based data handling remains relatively low and scales efficiently compared to the simulation stage for this experiment size.

4.5 Data Volume and Efficiency

The four simulation runs generated a total dataset comprising approximately $4 \times 3.3\text{M} \times 10 \approx 132$ million data points, culminating in 90 MB of final data stored as `.npy` arrays in this test. PUMLE effectively processes the initial JSON outputs into compact binary arrays containing only the target variables (pressure, Sw, Sg).

The framework supports generating Golden Layer arrays in either `.npy` or `.zarr` format:

- **`.npy`:** Offers simplicity and efficient I/O within the Python/NumPy ecosystem, benefiting ML frameworks. However, it requires loading the entire array into memory, hindering subset querying, and lacks embedded metadata and standard compression.
- **`.zarr`:** Provides chunking (enabling efficient partial reads/writes), built-in compression options, and better scalability for cloud or parallel environments. It is generally

more suitable for very large datasets where memory constraints or selective access are concerns.

While `.npy` was used for the performance results reported here due to its simplicity, the availability of Zarr within PUMLE provides a crucial alternative for addressing the limitations of `.npy` in larger-scale applications and improving adherence to FAIR data principles (e.g., by potentially storing metadata alongside array chunks). Traceability for both formats currently relies on the external database and file/path naming conventions.

4.6 Infrastructure Considerations, Limitations, and Future Outlook

All computations were performed on a single laptop, sufficient for demonstrating PUMLE’s workflow and the low relative overhead of its data handling components. However, scaling to realistic CCS research scenarios would necessitate more powerful infrastructure and addressing several limitations of the current implementation.

- **HPC/Cloud Deployment:** The framework’s design facilitates deployment on HPC clusters or cloud platforms. Parallel simulation execution can leverage multiple nodes, and cloud storage (AWS S3) provides a scalable repository (Figure 10). Integration with cloud ML platforms is feasible (SHOKOUHI *et al.*, 2021).
- **Scalability Bottlenecks:**
 - *Simulation Time:* MRST simulation remains the primary bottleneck.
 - *Memory Usage (Array Consolidation):* The current in-memory array consolidation in `arrays.py` presents a significant potential memory bottleneck for large grids/timesteps. Optimization using chunked processing (potentially facilitated by writing directly to Zarr) or memory-mapping is crucial for large-scale applicability.
 - *Cloud Transfer:* S3 uploads may require optimization (e.g., parallel transfers via boto3 Transfer Manager) for numerous large files.
- **Traceability and Reproducibility:** The current DB+naming convention for traceability is functional but lacks robustness for complex projects. Future work should prioritize:
 - Enhancing the database schema to store comprehensive parameters and link explicitly to data locations (paths/URIs).
 - Implementing robust data versioning using tools like DVC to track code, parameters, and data lineage.

- Leveraging Zarr’s capability to store metadata alongside array chunks, or exploring HDF5 for similar benefits.
- **Error Handling and Robustness:** Current error handling is basic. Production use requires more specific exception handling, potential retries for operations like cloud uploads, and detailed status logging in the database.
- **Extensibility and Deployment:** Integrating new simulators currently requires significant modification. Abstracting the simulator interface would improve modularity. Containerization using Docker is supported via an included **Dockerfile**, facilitating deployment standardization and reproducibility across different systems. Future work could involve optimizing the Docker image or providing pre-built images. Integration with HPC workload managers (Slurm) would enhance operational efficiency.

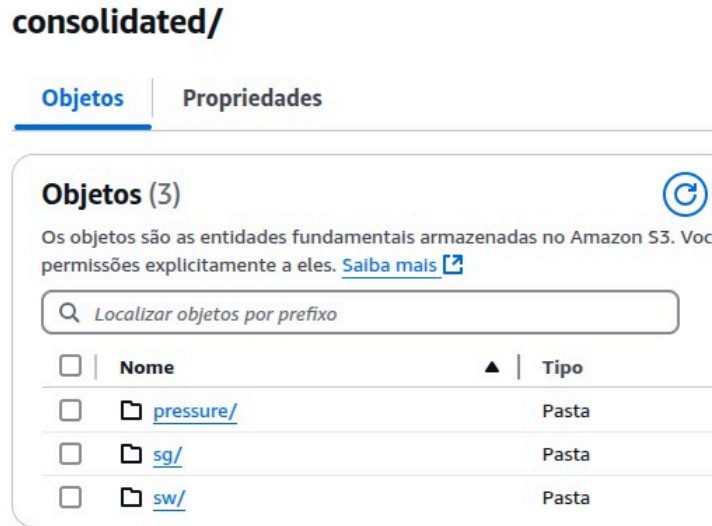


Figure 10: AWS S3 cloud storage artifacts showing organized folders for pressure and saturation consolidated data.

Future scientific expansions could involve integrating more complex physics (geomechanics, reactive transport (NGHIEM *et al.*, 2009)) into the simulation module. While increasing Bronze layer compute times, the PUMLE data pipeline structure should accommodate the outputs, with its relative overhead expected to remain small.

Overall, while the simulation is the main constraint, PUMLE provides an efficient workflow foundation with options for scalable data formats (.npz, .zarr) and standardized deployment (Docker). Addressing the identified limitations in memory scaling during consolidation, traceability robustness, and error handling is key to realizing its full potential for large-scale, reproducible data-driven CCS research.

5 CONCLUSION

This monograph introduced PUMLE, an integrated framework designed to manage the lifecycle of datasets generated from numerical simulations of CO₂ geological storage. Addressing the challenges of handling large simulation outputs, PUMLE orchestrates parameter definition, parallel simulation execution using MRST, parsing of raw JSON outputs, preparation of in-memory data structures, consolidation into analysis-ready 4D arrays (supporting both `.npy` and `.zarr` formats), and persistent storage locally or on AWS S3.

By adopting a layered workflow, the framework isolates the computationally intensive simulation stage. Performance evaluations confirmed that the MRST simulation dominates runtime, while the PUMLE data processing pipeline (parsing, array generation, storage) contributes minimal overhead (approx. 1.4% for the tested case using `.npy`). The framework successfully produces consistent, structured arrays representing pressure and saturation fields, suitable as input for ML tasks. Traceability is currently managed via an SQLite database and structured file/path naming, and deployment is facilitated by Docker support.

However, critical analysis identified areas for future enhancement. The current array consolidation method faces memory scalability challenges for large datasets, although using Zarr output could potentially mitigate this if consolidation writes directly to chunks. The traceability mechanism, while functional, could be made more robust through explicit data versioning and richer metadata integration (potentially leveraging Zarr’s capabilities). Improving error handling and optimizing cloud interactions are also key for operational use. While `.npy` offers simplicity, the framework’s support for `.zarr` provides a path towards better handling of large-scale data and metadata.

Based on PUMLE’s implementation and evaluation, the following future directions are recommended:

- **Enhance Scalability and Efficiency:** Optimize the array consolidation step for memory efficiency, potentially by writing directly to chunked Zarr format instead of creating full NumPy arrays in memory first. Perform comparative benchmarks for `.npy` vs. `.zarr` I/O performance in ML data loading scenarios. Optimize cloud data transfers.
- **Improve Reproducibility and Traceability:** Integrate robust data versioning (e.g., DVC) to link code, parameters, and data artifacts. Enhance the database schema to store comprehensive parameters and direct links (paths/URIs) to Golden Layer arrays. Fully leverage Zarr’s capability to embed essential metadata within the array store itself.

- **Enable Direct ML Integration:** Provide concrete examples and utility functions for loading PUMLE’s Golden Layer arrays (both `.npy` and `.zarr`) into PyTorch/TensorFlow for specific ML tasks (e.g., CNNs, PINNs (SHOKOUI *et al.*, 2021)). Demonstrate a baseline ML model trained on PUMLE-generated data.
- **Increase Framework Modularity:** Refactor simulation execution components to create a more abstract interface, facilitating the integration of alternative reservoir simulators.
- **Strengthen Deployment and Operations:** Implement more granular error handling and status reporting. Leverage and potentially optimize the existing Docker containerization support for simplified setup and consistent execution. Explore integration with HPC workload managers (Slurm).
- **Expand Community Engagement:** Following data warehousing and open science principles (KIMBALL, 1996; KLEPPMANN, 2017), investigate publishing benchmark datasets generated by PUMLE (potentially in Zarr format) or sharing the framework itself to foster collaboration.

Pursuing these directions will mature PUMLE into a more robust, scalable, and user-friendly platform, significantly strengthening the bridge between CCS simulation and advanced data-driven modeling.

REFERENCES

- GCCSI. **Global Status of CCS 2024: Collaborating for a Net-Zero Future**. [S.l.], 2024.
- AMINU, M. D.; NABAVI, S. A.; ROCHELLE, C. A.; MANOVIC, V. A review of developments in carbon dioxide storage. **Applied Energy**, v. 208, p. 1389–1419, 2017. ISSN 0306-2619. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306261917313016>.
- CHOUDHARY, P. **Carbon Capture and Storage Program(CCSP)**. [S.l.], 2016.
- DZIEJARSKI, B.; KRZYŻYŃSKA, R.; ANDERSSON, K. Current status of carbon capture, utilization, and storage technologies in the global economy: A survey of technical assessment. **Fuel**, v. 342, p. 127776, 2023. ISSN 0016-2361. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0016236123003897>.
- Norwegian Petroleum Directorate. **Sleipner Vest CO2 Disposal – The first Industrial Scale Project for CO2 Storage**. [S.l.], 2000.
- Northern Lights JV DA. **Northern Lights: CO2 transport and storage**. 2024.
- HARP, D.; ONISHI, T.; CHU, S.; CHEN, B.; PAWAR, R. Development of quantitative metrics of plume migration at geologic co₂ storage sites. **Greenhouse Gases: Science and Technology**, v. 9, n. 4, p. 687–702, 2019.
- FAN, M.; LU, D.; LIU, S. A deep learning-based direct forecasting of co₂ plume migration. **Geoenery Science and Engineering**, v. 221, p. 211363, 2023. ISSN 2949-8910. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2949891022000513>.
- BACHU, S. Co₂ storage in geological media: Role, means, status and barriers to deployment. **Progress in Energy and Combustion Science**, v. 34, n. 2, p. 254–273, 2008.
- (IPCC), I. P. on C. C. **Special Report on Carbon Dioxide Capture and Storage**. 2005. Cambridge University Press, Cambridge, UK.
- LIE, K.-A.; NILSEN, H. M.; ANDERSEN, O.; MØYNER, O. A simulation workflow for large-scale co₂ storage in the norwegian north sea. **Computational Geosciences**, Springer, v. 20, n. 3, p. 607–622, 2016.
- SPAN, R.; WAGNER, W. A new equation of state for carbon dioxide covering the fluid region from the triple-point temperature to 1100 k at pressures up to 800 mpa. **Journal of Physical and Chemical Reference Data**, v. 25, n. 6, p. 1509–1596, 1996.
- ROWE, A. M.; CHOU, J. C. S. Pressure-volume-temperature-concentration relation of aqueous nacl solutions. **Journal of Chemical and Engineering Data**, v. 15, n. 1, p. 61–66, 1970.
- DRIESNER, T. The system h₂o–nacl. part ii: Correlations for molar volume, enthalpy, and isobaric heat capacity from 0 to 1000°C, 1 to 5000 bar, and 0 to 1 x_{nacl}. **Geochimica et Cosmochimica Acta**, v. 71, n. 20, p. 4902–4919, 2007.

- MAO, S.; DUAN, Z. The viscosity of aqueous alkali-chloride solutions up to 623 K, 1,000 bar, and high ionic strength. **International Journal of Thermophysics**, v. 30, p. 1510–1523, 2009.
- KESTIN, J.; KHALIFA, H. E.; CORREIA, R. J. Tables of the dynamic and kinematic viscosity of aqueous NaCl solutions in the temperature range 20–150°C and the pressure range 0.1–35 MPa. **Journal of Physical and Chemical Reference Data**, v. 10, n. 1, p. 71–88, 1981.
- OLDENBURG, C. M.; SPYCHER, N.; PRUESS, K. CO₂ injection for storage in saline formations: Modelling and field demonstration aspects. **Greenhouse Gases: Science and Technology**, v. 6, p. 1–19, 2016.
- FAN, M. *et al.* Accelerating reservoir simulations of CO₂ sequestration with machine learning. **Computers & Geosciences**, v. 132, p. 74–86, 2019.
- HOSSEINI, S. A.; NICOT, J.-P.; ZEIDOUNI, M.; SOLANO, S.; HOVORKA, S. D. Reservoir modeling for CO₂ sequestration in response to field tests. **Environmental Science & Technology**, v. 47, p. 14382–14389, 2013.
- ZHANG, K.; WU, Y.-S.; DING, D.-S. Fully coupled modeling of geomechanics and multiphase flow for CO₂ injection and sequestration. **International Journal of Rock Mechanics and Mining Sciences**, v. 78, p. 86–95, 2015.
- WHITE, M. D.; ESPOSITO, R.; ZHANG, Y.; OLDENBURG, C. M. Comparison of coupled flow and geomechanical simulation approaches for CO₂ storage. In: DOE/NETL. **Proceedings of the 12th Annual Conference on Carbon Capture and Sequestration**. [S.l.], 2014.
- KUMAR, V.; BRYANT, S.; LAKE, L. Impact of heterogeneity on CO₂ storage capacity and plume shape. **Society of Petroleum Engineers Journal**, v. 15, p. 241–251, 2010.
- OBMANUEL, M.; HU, Y.; BATTIATO, I. Influence of topography on CO₂ plume migration in heterogeneous formations. **International Journal of Greenhouse Gas Control**, v. 63, p. 345–356, 2017.
- NGHIEM, L. X. *et al.* Simulation of geomechanics for CO₂ sequestration in saline aquifers and depleted oil/gas reservoirs. **SPE Journal**, v. 14, p. 323–338, 2009.
- SHOKOUHI, P.; KUMAR, V.; PRATHIPATI, S.; HOSSEINI, S. A.; GILES, C. L.; KIFER, D. Physics-informed deep learning for prediction of CO₂ storage site response. **Journal of Contaminant Hydrology**, v. 241, p. 103835, 2021.
- ZHANG, J.; GAO, Q.; DURLOFSKY, L. J. Reduced-order modeling for CO₂ sequestration in heterogeneous reservoirs using polynomial chaos expansions. **Water Resources Research**, v. 52, p. 73–92, 2016.
- TANG, M.; JIANG, X.; WU, Y.; ZHANG, J. Deep-learning-based approach to expedite multiphase flow predictions in carbon capture and storage. **Energy & Fuels**, v. 35, n. 17, p. 14085–14096, 2021.
- KIMBALL, R. **The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses**. New York: John Wiley & Sons, 1996.

KLEPPMANN, M. **Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems**. [S.l.]: O'Reilly Media, 2017.

PAWAR, R. J.; BROMHAL, G. S.; CHU, S.; DILMORE, R. M.; OLDENBURG, C. M.; STAUFFER, P. H. Rapid machine learning-based forecasting of co₂ saturation distribution for uncertainty quantification in geologic carbon storage. **International Journal of Greenhouse Gas Control**, v. 85, p. 106–118, 2019.

AMPOMAH, W.; BALCH, R. S.; CATHER, M. E.; ROSE-COSS, D.; DAI, Z.; HEATH, J. E.; MOZLEY, P. S. Co₂ storage and enhanced oil recovery: Approaches to data integration, analysis, and modeling for residual oil zones in the permian basin. In: SPE. **Society of Petroleum Engineers**. [S.l.], 2017.

PRATX, G. *et al.* **Large-scale data pipelines for geoscience simulations: Best practices and emerging solutions**. 2018. Preprint, arXiv:1804.00000. Accessed: YYYY-MM-DD.

ZHONG, X. *et al.* Forecasting of co₂ plume migration using conditional deep convolutional generative adversarial network (cdc-gan). 2019.

FAN, M.; WANG, H.; HOSSEINI, S. A.; LU, D. A deep learning-based direct forecasting of co₂ plume migration. **Computers & Geosciences**, 2022.

CHAPMAN, B.; LAGRONE, J. Openmp. In: _____. **Encyclopedia of Parallel Computing**. Boston, MA: Springer US, 2011. p. 1365–1371. ISBN 978-0-387-09766-4. Disponível em: https://doi.org/10.1007/978-0-387-09766-4{_}.

EATON, J. W.; BATEMAN, D.; HAUBERG, S.; WEHBRING, R. **GNU Octave version 8.4.0 manual: a high-level interactive language for numerical computations**. [S.l.], 2023. Disponível em: <https://www.gnu.org/software/octave/doc/v8.4.0/>.

APPENDICES

APPENDIX A - Example `setup.ini` Configuration File

The PUMLE framework utilizes an INI-style configuration file, typically named `setup.ini`, to define the parameters for a simulation run or a batch of runs. This appendix details the sections and parameters found in a typical configuration file, explaining their purpose and providing example values used in this work.

File Content Example

Below is an example of the `setup.ini` file content used for simulations in this monograph:

```
[Paths]
pumle_root = pumle
pumle_results = data_lake/bronze_data

[Pre-Processing]
case_name = GCS01
file_basename = db_sim
model_name = UNISIM-I-D

[Grid]
file_path = grid
repair_flag = True

[Fluid]
pres_ref = 35
temp_ref = 95.15
cp_rock = 4e-5
srw = 0.11
src = 0.21
pe = 5
xnacl = 0.1
rho_h2o = 1000

[Initial Conditions]
sw_0 = 1.0

[Boundary Conditions]
type = pressure

[Wells]
co2_inj = 1.5e5
wells_list = NA1A: (38, 36, 6, 11)

[Schedule]
```

```

injection_time = 15
migration_time = 30
injection_timesteps = 5
migration_timesteps = 5

[EXECUTION]
octave = /usr/bin/octave
mrst_root = /home/luiz/Downloads/mrst-2024b/

[SimNums]
sim_id = 1

```

Listing 1: .INI sample file

Parameter Descriptions

The parameters within the `setup.ini` file are organized into sections:

[Paths]

Defines directory paths used by the PUMLE framework.

pumle_root

Path to the root directory of the PUMLE source code. Example: `pumle`.

pumle_results

Path where the initial simulation results (Bronze Layer data) should be stored.
Example: `data_lake/bronze_data`.

[Pre-Processing]

Parameters related to case identification and model setup.

case_name

A user-defined name for the specific simulation case. Example: `GCS01`.

file_basename

A base name used for generating output filenames, aiding organization. Example:
`db_sim`.

model_name

Identifier for the reservoir grid model being used. Example: `UNISIM-I-D` (referencing the model used in Section [4](#)).

[Grid]

Parameters related to the reservoir grid definition.

file_path

Relative or absolute path to the grid file(s). Example: `grid`.

repair_flag

Boolean flag indicating whether to attempt automatic repairs on the grid structure (e.g., ZCORN values) if issues are detected. Example: `True`.

[Fluid]

Defines the properties of the fluids and rock involved in the simulation.

pres_ref

Reference pressure, often corresponding to initial reservoir conditions. Unit: MPa. Example: 35 (corresponding to 35 MPa as used in Section 4.2).

temp_ref

Reference temperature, often corresponding to initial reservoir conditions. Unit: K or °C (ensure consistency with simulation code, 95.15 K used in Section 4.2, while Appendix 6 uses °C). Example: 95.15.

cp_rock

Rock compressibility. Unit: bar. Example: 4e-5 (corresponds to 4×10^{-5} bar).

srw

Residual saturation of the wetting phase (brine). Dimensionless. Example: 0.11.

src

Residual saturation of the non-wetting phase (CO₂). Dimensionless. Example: 0.21.

pe

Capillary entry pressure, related to the maximum capillary pressure ($p_c = p_e s_w^{-1/2}$). Unit: kPa. Example: 5 (corresponds to 5 kPa).

xnac1

Mass fraction of NaCl used to calculate brine properties (density, viscosity). Dimensionless. Example: 0.1 (corresponds to 10% salinity as used in Section 4.2).

rho_h2o

Reference density of pure water, often used in property correlations. Unit: kg/m³. Example: 1000.

[Initial Conditions]

Specifies the state of the reservoir at the beginning of the simulation.

sw_0

Initial water (brine) saturation throughout the model domain. Dimensionless. Example: 1.0 (representing a reservoir initially fully saturated with brine).

[Boundary Conditions]

Defines how the simulation domain interacts with its surroundings.

type

Specifies the type of boundary condition applied to the model's external faces (vertical and lateral). Options are typically constant **pressure** or zero **flux** (no flow). Example: **pressure**.

[Wells]

Parameters controlling the injection wells.

co2_inj

Volumetric injection rate of CO₂, usually specified at standard or surface conditions. Unit: m³/year. Example: 1.5e5 (corresponds to 1.5×10^5 m³/year as used in Section 4.2).

wells_list

Specifies the location of well perforations within the grid using logical indices (I, J, K_min, K_max). Multiple wells can be defined. Example: NA1A: (38, 36, 6, 11).

[Schedule]

Defines the timing and time-stepping strategy for the simulation periods.

injection_time

Duration of the CO₂ injection period. Unit: years. Example: 15 (corresponds to 15 years as used in Section 4.2).

migration_time

Duration of the post-injection period (migration and monitoring). Unit: years. Example: 30.

injection_timesteps

Number of simulation time steps to take during the injection period. Example: 5.

migration_timesteps

Number of simulation time steps to take during the migration period. Example: 5.

[EXECUTION]

Specifies paths to external software required by PUMLE.

octave

Full path to the Octave executable (or MATLAB if adapted). Required for running MRST scripts. Example: `/usr/bin/octave`.

mrst_root

Full path to the root directory of the MRST installation. Example: `/home/luiz/Downloads/mrst`

[SimNums]

Parameters related to simulation identification, particularly for batch runs.

sim_id

An identifier for the specific simulation run, useful when multiple simulations are executed. Example: 1.

APPENDIX B - Fluid Property Models in co2lab3DPUMLE

This appendix presents the key mathematical formulations implemented in the co2lab3DPUMLE simulation, with focus on the physical properties of the CO₂-brine system in deep saline aquifers. The formulations are based on literature models and implemented in the simulation code to compute fluid density, viscosity, compressibility, and saturation functions.

Reservoir Conditions

The pressure p and temperature T at a given reservoir depth h are modeled using linear gradients:

$$\begin{aligned} p(h) &= p_0 + G_p h \\ T(h) &= T_0 + G_T h \end{aligned}$$

where:

- $p_0 = 0.1$ MPa is the atmospheric reference pressure.
- $T_0 = 4$ °C is the seabed temperature.
- $G_p = 10$ MPa km⁻¹ is the hydrostatic pressure gradient.
- $G_T = 10$ MPa km⁻¹ is the geothermal gradient.

Brine Density

The brine density ρ_w is interpolated between pure water and sodium chloride (NaCl) as:

$$\rho_w = \rho_{\text{H}_2\text{O}} X_{\text{H}_2\text{O}} + \rho_{\text{NaCl}} X_{\text{NaCl}}$$

The density of pure NaCl liquid is estimated using (DRIESNER, 2007):

$$\begin{aligned} \rho_{\text{NaCl}}^0 &= \frac{m_0}{m_1 + m_2 T + m_3 T^2} \\ c_{\text{NaCl}} &= m_4 + m_5 T \\ \rho_{\text{NaCl}} &= \frac{\rho_{\text{NaCl}}^0}{1 - 0.1 \log(1 + 10 P c_{\text{NaCl}})} \end{aligned}$$

with coefficients:

$$\begin{aligned} m_0 &= 58443, & m_1 &= 23.772, & m_2 &= 0.018639, \\ m_3 &= -1.9687 \times 10^{-6}, & m_4 &= -1.5259 \times 10^{-5}, & m_5 &= 5.5058 \times 10^{-8} \end{aligned}$$

Brine Viscosity

Following (MAO; DUAN, 2009), the brine viscosity μ_w is derived from the relative viscosity μ_r and water viscosity $\mu_{\text{H}_2\text{O}}$:

$$\mu_w = \mu_r \cdot \mu_{\text{H}_2\text{O}}$$

The molality m of brine is computed as:

$$m = \frac{1000X_{\text{NaCl}}}{M_{\text{NaCl}}(1 - X_{\text{NaCl}})}$$

Relative viscosity:

$$\begin{aligned} A &= a_0 + a_1T + a_2T^2 \\ B &= b_0 + b_1T + b_2T^2 \\ C &= c_0 + c_1T \\ \log(\mu_r) &= Am + Bm^2 + Cm^3 \end{aligned}$$

Water viscosity:

$$\log(\mu_{\text{H}_2\text{O}}) = \sum_{i=1}^5 d_i T^{i-3} + \sum_{i=6}^{10} d_i \rho_{\text{H}_2\text{O}} T^{i-8}$$

with constants a_i, b_i, c_i, d_i from (MAO; DUAN, 2009).

Capillary Pressure

Capillary pressure is modeled using an inverse square-root relation:

$$p_c(s_w) = p_e s_w^{-1/2}$$

where p_e is the entry pressure, and s_w is the water saturation.

Relative Permeability

Relative permeabilities are scaled to residual saturations s_{rw} and s_{rc} as:

$$\begin{aligned}k_{rw}(s) &= k_{rw} \left(\max \left(\frac{s - s_{rw}}{1 - s_{rw}}, 0 \right) \right) \\k_{rg}(s) &= k_{rg} \left(\max \left(\frac{s - s_{rc}}{1 - s_{rc}}, 0 \right) \right)\end{aligned}$$

These mathematical models ensure that fluid and rock properties are grounded in realistic physical behavior at the scale of CO₂ sequestration simulations in Brazilian reservoirs.

APPENDIX C - Code Structure

All source code and documentation related to the PUMLE system can be found in the project's official repository under the main branch: [PUMLE](#). The following directory structure summarizes the core components of the project, including simulation scripts, configuration files, and the data engineering pipeline responsible for orchestrating simulation, post-processing, and cloud upload:

```
benchmark/  
  unisim-1-d/  
    KR_0.INC  
    PETRO_0.INC  
    PVT_0.INC  
    SO.INC  
    UNISIM_I_D_ECLIPSE.DATA  
  
simulation/  
  co2lab3DPUMLE.m  
  simulation.cpp  
  
src/  
  pumle/  
    __init__.py  
    arrays.py  
    cloud_storage.py  
    db.py  
    ini.py  
    mat_files.py  
    metadata.py  
    parameters.py  
    parameters_variation.py  
    paths.py  
    pumle.py  
    sim_results_parser.py  
  
main.py  
simulation_script.sh
```

Listing 2: Code artifacts structure.

APPENDIX D - PUMLE Article Publication

The author has published the following conference paper related to PUMLE:

- Santos, L. F. C. dos and Oliveira, G. P. (2025). *Generation-to-Storage Data Framework for Large-Scale Carbon Sequestration Simulation Datasets*. In: *Proceedings of the XX Escola Regional de Banco de Dados (ERBD 2025)*. [<https://doi.org/xxx/xxx>](https://doi.org/xxx/xxx)