Particle Detector Simulation - PHYS30762 C++ Project

Yupeng Fan

11069686

School of Physics and Astronomy

University of Manchester

05 2025

May 10, 2025

Abstract

This project develops a particle detector simulator for high-energy physics experiments, incorporating sub-detectors as in CMS. It employs classes for particles, sub-detectors, and the detector system, with particles and sub-detectors acting as wrappers for energy and momentum calculations. The detector class oversees the components, conducts simulations, and outputs results to both the console and a file for analysis.

1 Introduction

The CMS experiment at CERN employs sophisticated particle detectors to probe fundamental physics. This project simulates a simplified CMS-like detector, comprising sub-detectors such as trackers, calorimeters, hadronic calorimeters, and muon chambers, and particles like photons, electrons, neutrinos, hadrons, and muons. Key features include randomized energy detection, particle type inference, and MET calculations.

The simulation incorporates several key physical models. The measured energy E_{measured} from

a sub-detector is calculated as:

$$E_{\text{measured}} = \begin{cases} 0 & \text{if } \text{rand}_{\text{uniform}} > \epsilon, \\ \max(0, E_{\text{true}} + \mathcal{N}(0, \sigma)) & \text{otherwise,} \end{cases} \tag{1}$$

where ϵ is the detection efficiency, $\sigma = E_{\text{true}} \cdot \text{resolution}$ is the Gaussian noise standard deviation, and $\mathcal{N}(0, \sigma)$ represents random fluctuations.

The missing transverse energy (MET) is computed to account for undetected particles like neutrinos:

$$\text{MET} = \sqrt{\left(\sum p_x\right)^2 + \left(\sum p_y\right)^2}, \tag{2}$$

where $\sum p_x$ and $\sum p_y$ are the total transverse momenta of all particles.

Energy-momentum consistency is validated using the relativistic relation:

$$E = \sqrt{p^2 + m^2}, \tag{3}$$

where $p = \sqrt{p_x^2 + p_y^2 + p_z^2}$ is the momentum magnitude and m is the particle mass.

This report details the code design, implementation, results, and potential enhancements, emphasizing advanced C++ techniques and simulation accuracy.

2 Class

2.1 Class Hierarchy

The simulation is structured around a robust class hierarchy. The `Detector` class manages multiple `SubDetector` objects, such as `Tracker`, `Calorimeter`, `HadronicCalorimeter`, and `MuonChamber`. The `Particle` abstract base class supports derived classes like `Photon`, `Electron`, `Neutrino`, `Hadron`, and `Muon`, each containing a `FourMomentum` object to store energy and momentum. This hierarchy ensures polymorphism, allowing flexible addition of new detector components or particle types.

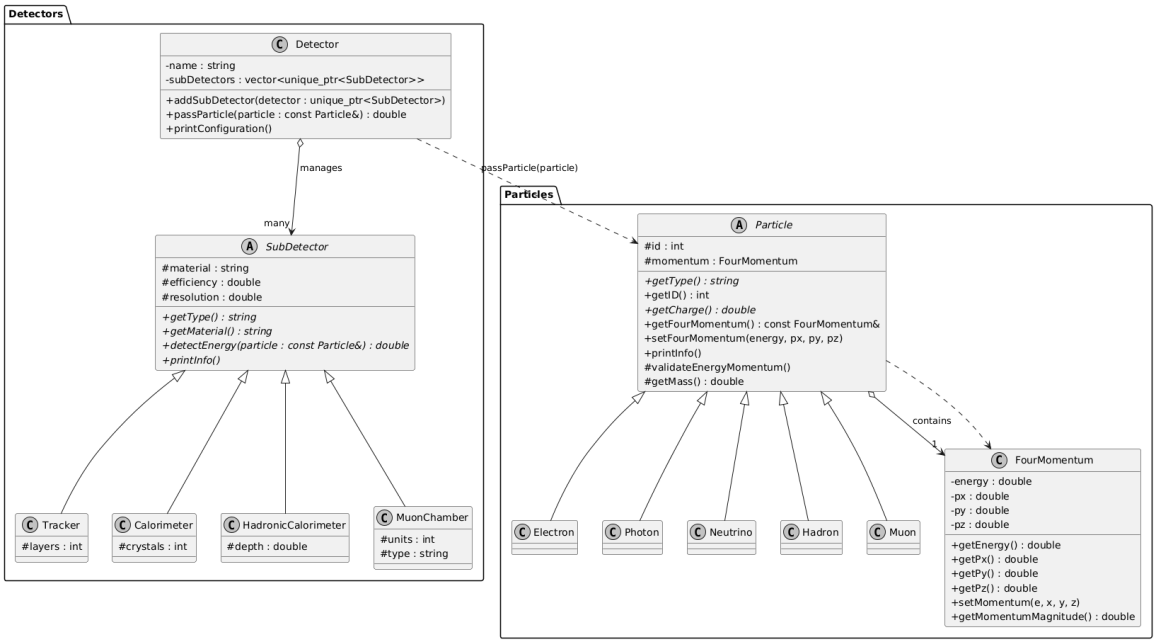

Figure 1: UML diagram of the particle detector simulation class hierarchy.

2.2 Discussion of Each Class and Design Reasoning

2.2.1 Detector Class

The `Detector` class is the core of the simulation, responsible for organizing the interaction between particles and sub-detectors. It is designed to manage a dynamic list of sub-detectors, allowing users to configure the detector setup flexibly, a key requirement inspired by the CMS experiment's modular design.

Unique pointers manage memory safely and prevent leaks. They also allow dynamic addition of sub-detectors. The `passParticle` method simulates particle interactions. It iterates over sub-detectors, collects measurements, and infers particle types. The `printConfiguration` method improves usability. It provides a clear overview of the detector setup and helps with debugging and user interaction.

2.2.2 SubDetector Class

The `SubDetector` class is an abstract base for every sub-detector type. It offers a shared interface and common functions. With this setup, each sub-detector works in the same framework but can add its own detection method.

2.2.3 Tracker Class

The `Tracker` class is a subclass of `SubDetector`. It models a tracking detector that detects charged particles. The design follows the CMS tracker. It records the paths of particles such as electrons and muons.

The `Tracker` detects only charged particles (`particle.getCharge() != 0`). Neutral particles such as photons leave no tracks. The `layers` attribute copies the layered structure of real trackers, and `material` (for example, silicon) shows the physical make-up. The `detectEnergy` method adds randomness. This feature gives realistic energy readings and helps simulate detector flaws.

2.2.4 Calorimeter Class

`Calorimeter` models an electromagnetic calorimeter, which measures the energy of neutral particles such as photons and charged particles such as electrons.

The `crystals` stores the number of scintillating crystals (for example, lead tungstate), a main part of the CMS electromagnetic calorimeter. The function `detectEnergy` detects neutral particles. This choice shows the calorimeter's task of measuring photon and electron energy. The random step makes each reading change in a realistic way, matching real calorimeter behavior.

2.2.5 HadronicCalorimeter Class

`HadronicCalorimeter` models a hadronic calorimeter. It measures hadron energy by stopping them in dense material.

The `depth` gives the calorimeter thickness. Thick layers absorb hadronic showers. The class targets hadrons only. The function `detectEnergy` tests whether the particle is a `Hadron`. This focus keeps energy readings accurate for hadrons in a CMS-like detector.

2.2.6 MuonChamber Class

The `MuonChamber` class models a muon detection system, designed to identify muons that penetrate other sub-detectors.

The `chamberType` (for example, DT or CSC) and `units` show the range of muon detection methods in CMS. The class detects muons only, which keeps the simulation true to the way muons pass through other sub-detectors without being detected.

2.2.7 Particle Class

The `Particle` class is an abstract base class for all particle types, providing a common interface for particle properties. The `id` gives each particle a unique label, while `FourMomentum momentum` stores its kinematic data. The abstract design lets you add new particle kinds, like pions or neutrinos, by adding new classes only. The `getCharge` method lets sub-detectors decide if they can see the particle.

2.2.8 FourMomentum Class

The `FourMomentum` class encapsulates a particle's energy and momentum, providing a clean interface for kinematic calculations.

This class is designed for simplicity and efficiency, storing four-momentum components directly. It supports MET calculations (Equation (2)) and energy-momentum validation (Equation (3)), ensuring physical accuracy in the simulation. The lack of validation in the constructor is intentional, as consistency checks are handled in `main.cpp` via `validateEnergyMomentum`.

2.2.9 Particle Derived Classes (Photon, Electron, Neutrino, Hadron, Muon)

The `Photon`, `Electron`, `Neutrino`, `Hadron`, and `Muon` classes inherit from `Particle`, each representing a specific particle type with appropriate physical properties.

Listing 1: Example: Definition of Photon class

```
class Photon : public Particle
{
public:
  Photon(int id = 0, double energy = 0.0);
  std::string get_type() const override { return "Photon"; }
  double get_charge() const override { return 0.0; }

protected:
  double get_mass() const override;
};
```

These classes keep particle properties. They store charge and mass, which `validateEnergyMomentum` uses. Sub-detectors read the particle type and charge. This lets `Calorimeter` spot photons and

`Tracker` spot electrons. The simplicity of these classes ensures that the simulation remains focused on detector behavior rather than complex particle physics.

3 Implementation

3.1 Simulation Workflow and Energy Measurement

The simulation workflow begins with a particle passing through the detector, where each sub-detector evaluates its interaction based on physical properties. The process involves two key aspects: particle type inference and energy measurement.

3.1.1 Particle Type Inference

Particle identification is performed by analyzing the detection patterns across sub-detectors. The detector iterates through its sub-detector collection, recording which detectors register a hit (e.g., tracker, calorimeter). Based on these patterns, the particle type is inferred using predefined rules. For instance, a hit only in the calorimeter suggests a photon, while hits in both the tracker and calorimeter indicate an electron. This mimics the CMS experiment's approach to particle identification, where detector signatures guide classification. The inference process is visually represented in Figure 2, which outlines the decision-making logic.

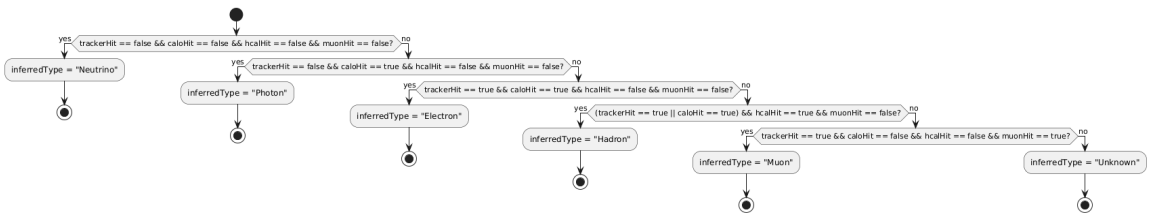

Figure 2: Flowchart of the particle inference process.

3.1.2 Energy Measurement

Energy measurement is a critical component of the simulation, designed to reflect the stochastic nature of real detectors. Each sub-detector implements the `detectEnergy` method, which calculates the measured energy based on the particle's true energy, detection efficiency, and resolution. The process uses the model in Equation (1). First, it makes a uniform random check to decide whether detection happens. Then it adds Gaussian noise to copy measurement uncertainty.

For example, the `Tracker` class measures energy for charged particles, incorporating randomization as shown below.

Listing 2: Energy measurement in Tracker

```cpp
double Tracker::detect_energy(const Particle &particle) const
{
  // Tracker detects charged particles (electrons, muons, hadrons)
  if (uniform(gen) > efficiency)
  {
    return 0.0; // No detection
  }

  if (particle.get_charge() != 0.0)
  {
    double energy = particle.get_four_momentum().get_energy();
    double sigma = energy * resolution;
    double measured_energy = energy + dist(gen) * sigma; // Gaussian
        noise
    return std::max(0.0, measured_energy);               // Ensure
        energy is non-negative
  }
  return 0.0; // Neutral particles (e.g., photons) not detected
}
```

The `efficiency` check ensures that not all particles are detected, reflecting real-world inefficiencies. The `resolution` parameter times the true energy sets the standard deviation of the Gaussian noise (`dist(gen) * sigma`). The code adds this noise to the true energy to get the measured value. A 60.00 GeV photon can then read 60.23 GeV, so the spread looks real. The `Detector` class sums the measured energies from all sub-detectors and gives one total for later analysis.

3.2 Additional Features

The simulation incorporates advanced features to enhance functionality, robustness, and user experience, including interactive input, randomized detection, and comprehensive error checking.

3.2.1 Interactive Input

The simulation provides an interactive interface for users to configure particle parameters. The process begins with a prompt, asking whether the user want to input particle parameters manually. Typing y/Y loads predefined default particles (e.g., a 60 GeV photon), while N/n allows manual input of particle types and kinematic properties. Each step, including the initial choice and subsequent particle parameter inputs, incorporates error checking to ensure data validity. Users can add multiple particles, confirming their inputs before the simulation starts. This workflow ensures flexibility and reliability.

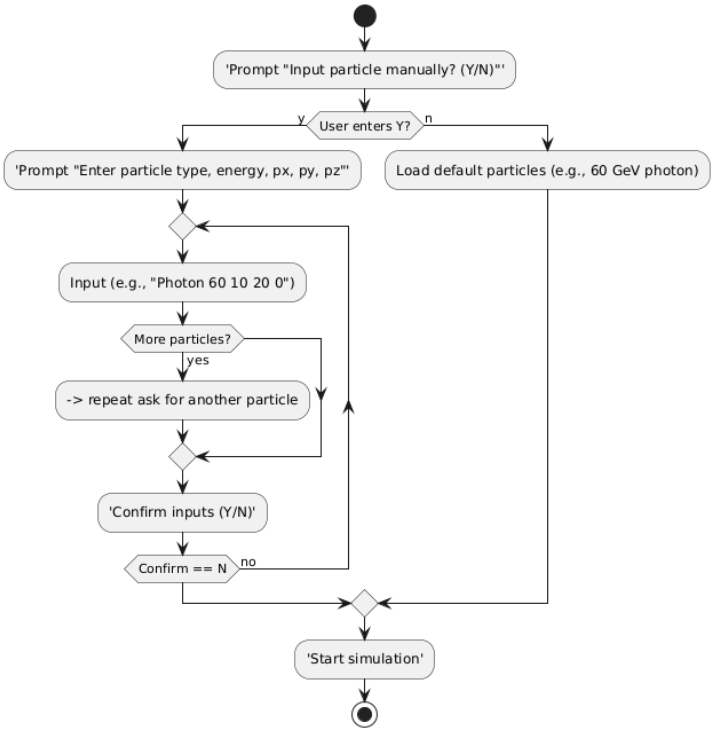

Figure 3: Flowchart of the interactive input process.

3.2.2 Randomized Detection

Randomization simulates the stochastic nature of particle detection. Each sub-detector applies a detection efficiency check using a uniform random distribution, followed by Gaussian noise addition based on the resolution, as per Equation (1). This results in realistic energy fluctuations, such as a 60.00 GeV photon measured as 60.23 GeV, enhancing the simulation's alignment with physical experiments.

3.2.3 Error Checking

Robust error-checking mechanisms ensure data integrity and physical consistency, integrated across different stages of the simulation. The process is divided into three distinct parts, as shown in Figure 4:

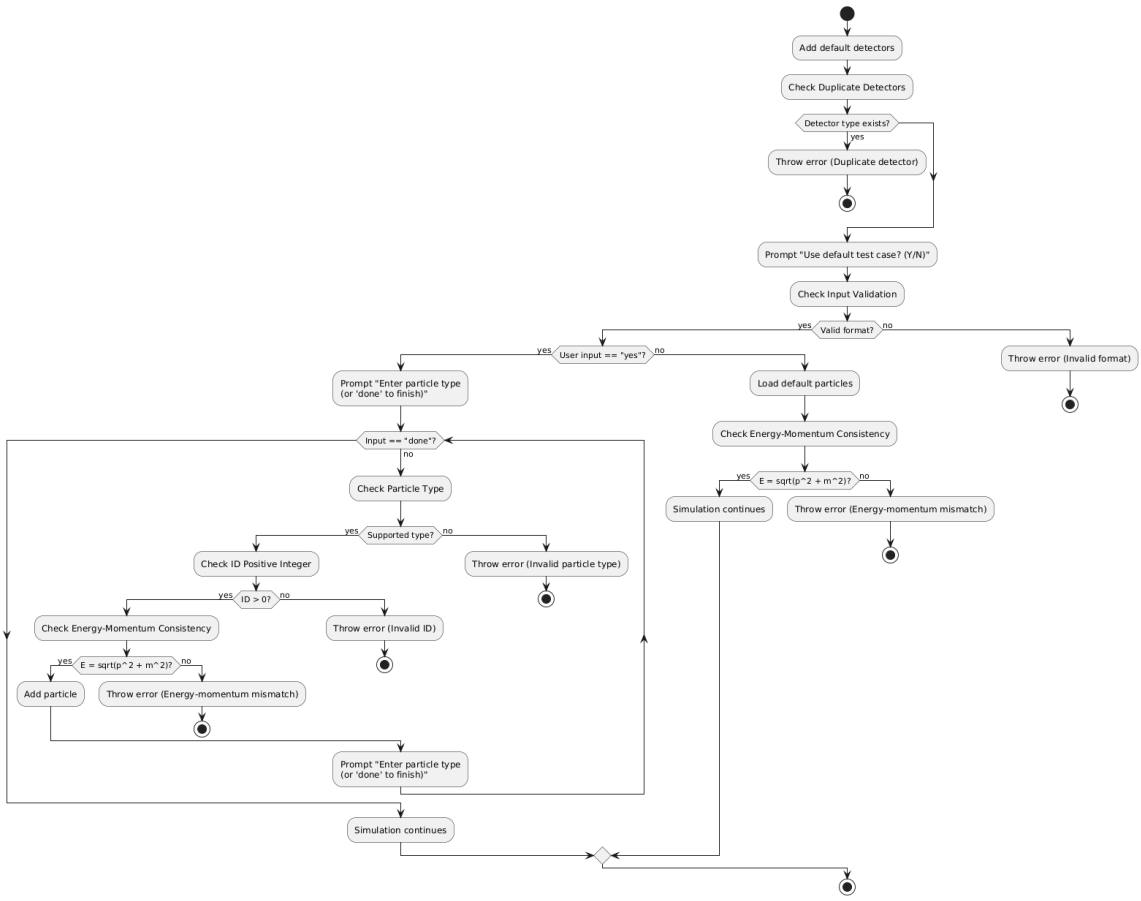

Figure 4: Flowchart of the error checking process, divided into three parts: Interactive Input, Default Detector, and Default Particle error handling.

Interactive Input Error Handling:

- Input Validation Check: Ensures inputs (e.g., valid momentum values) meet format requirements.

- ID Positive Integer Check: Confirms that the particle ID is a positive integer.

- Particle Type Validation Check: Verifies that the entered particle type (e.g., Photon, Electron) is supported.

Default Detector Error Handling:
- Duplicate Detectors Check: Prevents adding duplicate sub-detector types, enforced by the function ‘void Detector::addSubDetector‘ in ‘Detector.cpp‘, which throws an exception if a detector type already exists.

Default Particle Error Handling:
- Energy-Momentum Consistency Check: Verifies that a particle's energy matches its momentum and mass using Equation (3).

These steps maintain the simulation's reliability for complex studies.

4 Results

The simulation was tested with a CMS-like configuration, processing particles such as photons, electrons, neutrinos, hadrons, and muons. Summary outputs include true and measured energies, inferred particle types, and MET, which are also stored in a text file for further analysis.

4.1 Default Particle Configuration

The default configuration includes a set of particles with predefined energies and momenta. An example for a photon is shown below:

```
1       // Photon: E = 60 GeV
2       particles.push_back(std::make_unique<Photon>(1, 60.0));
3       particles.back()->set_four_momentum(60.0, 60.0 / std::sqrt(2),
           60.0 / std::sqrt(2), 0.0);
4       usedIDs.insert(1);
```

4.2 Simulation Output and Analysis

The simulation results are shown in the terminal output (Figure 5), which includes detailed particle interactions with the Simple-CMS detector. The detector configuration consists of a Tracker (efficiency 0.99, resolution 0.01), Calorimeter (efficiency 0.98, resolution 0.02), Hadronic

Calorimeter (efficiency 0.95, resolution 0.1), and two Muon Chambers (efficiency 0.99, resolution 0.05 each).

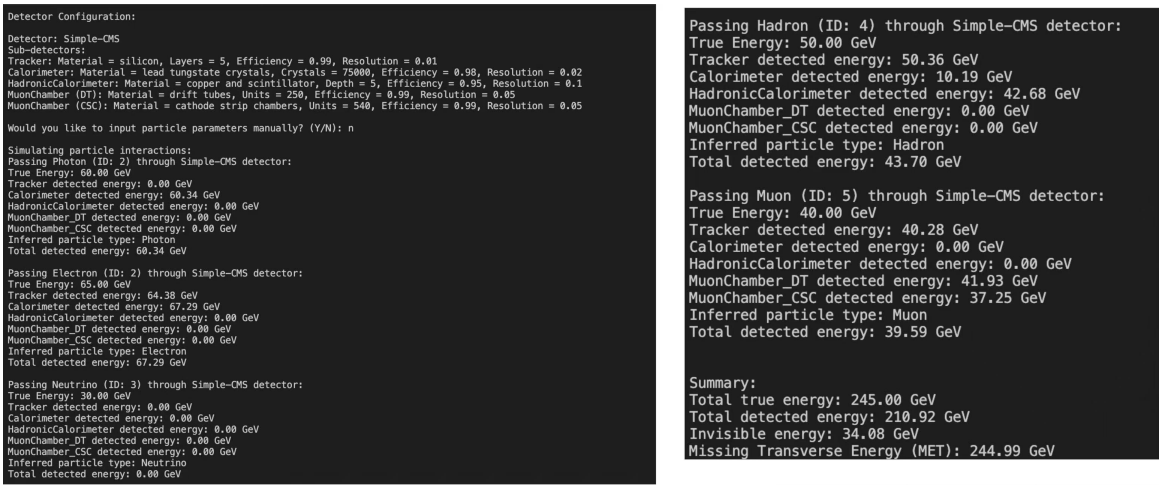

Figure 5: Terminal output of simulation results, showing particle interactions and summary metrics.

To understand the detection efficiency and its principles, we analyze the Photon (ID 1, true energy 60.00 GeV). It is detected solely by the Calorimeter at 60.34 GeV, with the Calorimeter having an efficiency of 0.98 and resolution of 0.02.

Detection Principle: Photons interact with the Calorimeter (lead tungstate crystals) via electromagnetic showers. A high-energy photon undergoes pair production or Compton scattering, creating a cascade of particles that deposit energy, which is converted into a measurable signal (e.g., scintillation light) [1].

Efficiency Analysis: The Calorimeter's efficiency of 0.98 indicates a 98% detection probability, attributed to its dense material maximizing shower capture. The 2% inefficiency may result from shower leakage or signal conversion losses. The photon was successfully detected, as expected.

Resolution and Noise: The resolution of 0.02 introduces uncertainty, modeled as Gaussian noise with $\sigma = 60.00 \cdot 0.02 = 1.20$ GeV. The measured energy (60.34 GeV) deviates by 0.34 GeV, within one standard deviation, reflecting stochastic shower fluctuations.

This demonstrates the Calorimeter's effectiveness in photon detection, with similar principles

applying to other sub-detectors.

5 Discussion and Conclusions

The simulation provides a robust framework for particle detection, accurately modeling interactions of particles like photons and muons in a CMS-like detector. The high efficiencies of the sub-detectors (e.g., Calorimeter at 0.98) and the calculated MET for default particles(244.99 GeV) reflect the simulation's alignment with real-world physics experiments.

Future improvements could enhance the simulation's realism and flexibility. Currently, the simulation uses `std::mt19937 gen` for randomization, a Mersenne Twister pseudo-random number generator, to model detection efficiencies and noise. While this provides good statistical uniformity, it has limitations: it lacks the physical realism of true stochastic processes in particle interactions and can introduce correlations in long sequences, potentially skewing results in large-scale simulations. Adopting more sophisticated randomization models, such as Monte Carlo methods used in GEANT4 [2], would better simulate complex scattering processes and energy losses, improving the accuracy of particle interaction modeling.

A specific area for improvement in the code is the `validateEnergyMomentum` function, used to check energy-momentum consistency (e.g., for a photon where a mismatch of 20.00 GeV vs. 34.641016 GeV was detected). This function currently performs a basic consistency check but does not account for relativistic effects across all particle types, potentially missing edge cases like ultrarelativistic particles or particles with zero rest mass. Enhancing this function to incorporate a relativistic energy-momentum relation—such as ensuring $E^2 - (p_x^2 + p_y^2 + p_z^2)c^2 = (mc^2)^2$ for all particles—would improve its robustness and ensure physical accuracy across diverse particle types.

6 Acknowledgements

I drew some inspiration from the CMS detector documentation at `https://cms.cern/detector`, particularly in incorporating the hadronic calorimeter into the simulation design. The code in this project is entirely my own work. In preparing this report, I used AI assistance solely for grammar and language polishing.

References

[1] CMS Collaboration. The cms experiment at the cern lhc. *JINST*, 3:S08004, 2008.

[2] S. Agostinelli et al. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003.