KØBENHAVNS UNIVERSITET

# PAT - Project

Mikkel Willén, bmq419

June 23, 2024

**Abstract**

This study aims to model and simulate critical mass phenomena within the Janus playground. Critical mass is a pivotal concept in nuclear physics, referring to the minimum amount of fissile material needed to sustain a nuclear chain reaction. The focus will be on three core problems: the calculation of critical mass for a given fissile material, the dynamics of a chain reaction, and the conditions for achieving and maintaining supercriticality.

The primary challenges include adapting continuous mathematical models of critical mass into a discrete format suitable for Janus, which only accepts integer inputs. The first model involves calculating the critical mass threshold for various fissile materials by simulating the number of neutrons produced and absorbed in the material. The second model examines the dynamics of a sustained chain reaction, focusing on the balance between neutron production and loss. The third model investigates the conditions necessary for achieving supercriticality, where the reaction becomes self-sustaining and grows exponentially.

These problems, while straightforward in theoretical physics, present significant implementation challenges within the constraints of the Janus playground. Specifically, we will investigate if Janus can accurately handle the reversibility of critical mass equations, maintaining consistency in forward and reverse simulations. The results will be validated through a series of controlled experiments, comparing the outputs against known physical data. Trace outputs of the simulations will be included in the appendix for verification purposes.

Should these problems prove too elementary or be resolved rapidly, additional, more complex critical mass scenarios will be introduced to further test the capabilities of Janus.

# Contents

# 1 Introduction

The study of nuclear physics encompasses a wide range of phenomena, among which the concept of critical mass holds significant importance. Critical mass refers to the minimum quantity of fissile material required to sustain a nuclear chain reaction. This concept is crucial for both civilian applications, such as energy generation in nuclear reactors, and military applications, such as the development of nuclear weapons. Understanding the conditions under which a fissile material achieves criticality is essential for safe and efficient nuclear technology design and operation.

In a nuclear chain reaction, fissile materials like Uranium-235 and Plutonium-239 undergo fission, releasing neutrons that can induce further fission in nearby atoms. For a self-sustaining chain reaction, each fission event must produce enough neutrons to continue the process. This delicate balance between neutron production and absorption determines whether the chain reaction will be subcritical, critical, or supercritical. Achieving critical mass means reaching a state where the reaction is self-sustaining, while surpassing it can lead to an exponentially increasing reaction rate, often resulting in an explosive energy release [1].

This project aims to explore and model the critical mass and related properties of fissile materials through a computational approach. By leveraging the Janus programming language, which supports reversible computing, the project simulates the conditions necessary for achieving critical mass. This includes calculating the critical radius and integrating iterative adjustments of mass and density to ensure the system reaches criticality.

The models developed in this project serve as foundational tools for understanding the physics behind nuclear chain reactions. These models not only provide insights into the behavior of fissile materials but also offer practical applications in nuclear reactor design and safety analysis. Through detailed simulations and iterative calculations, this project seeks to contribute to the broader field of nuclear physics by providing accurate and adaptable models for educational, research, and practical purposes.

# 2 Background

## 2.1 Critical mass

Critical mass is a fundamental concept in nuclear physics, referring to the minimum amount of fissile material needed to maintain a self-sustaining nuclear chain reaction. When a fissile material, such as Uranium-235 or Plutonium-239, undergoes fission, it releases neutrons. These neutrons can then induce further fission reactions in nearby fissile atoms, creating a chain reaction. For the chain reaction to be self-sustaining, each fission event must produce enough neutrons to sustain subsequent reactions.

In a nuclear chain reaction, the balance between neutron production and absorption is crucial. When a fissile nucleus undergoes fission, it typically releases two to three neutrons. These neutrons can either cause additional fissions, be absorbed by other materials (including the fissile material itself), or escape from the system. The concept of critical mass revolves around achieving a balance where the number of neutrons produced is sufficient to sustain the reaction.

If the amount of fissile material is below the critical mass, the chain reaction will not be self-sustaining. This is called **subcritical mass**. Neutrons are lost through absorption and escape, leading to a decrease in the reaction rate. At **critical mass**, the chain reaction becomes self-sustaining. Each fission event, on average, causes at least one more fission event, maintaining a steady rate of reaction. If the mass of the fissile material exceeds the critical mass, the chain reaction becomes **supercritical**. The reaction rate increases exponentially, potentially leading to an explosive release of energy.

The critical mass $M_c$ of a spherical mass of fissile material can be approximated using the following

equation:

$$M_c = \frac{4\pi R_c^3 \rho}{3}$$

where $R_c$ is the critical radius, and $\rho$ is the density of the fissile material.

The critical radius $R_c$ can be found using the neutron mean free path $\lambda$, the multiplication factor $k$ and the gemoetric buckling $B_g$ as follows:

$$R_c = \frac{\pi}{B_g}$$

For a sphere, the geometric buckling is given by:

$$B_g^2 = \frac{\pi^2}{R^2}$$

and the multiplication factor $k$, which must be equal to 1 for criticality, is defined as:

$$k = \eta \cdot \epsilon \cdot p \cdot f$$

where $\eta$ is the number of neutrons produced per absorption in the fuel, $\epsilon$ is the fast fission factor, $p$ is the resonance escape probability and $f$ is the thermal utilization factor.

Several factors influence the critical mass of a fissile material. Different fissile materials have different neutron production and absorption characteristics. Uranium-235 and Plutonium-239 are commonly used due to their favorable properties. The geometry and density of the fissile material affect neutron escape. A more compact shape (e.g., a sphere) and higher density reduce neutron loss, lowering the critical mass. Surrounding the fissile material with neutron reflectors can bounce escaping neutrons back into the material, effectively reducing the critical mass. Moderators can slow down fast neutrons, making them more likely to induce fission.

Understanding and calculating critical mass is essential in both civilian and military applications. In nuclear reactors, maintaining a controlled, self-sustaining chain reaction is crucial for steady energy production. Reactor design ensures the system remains at or near critical mass under operational conditions. In nuclear weapons, achieving a supercritical mass momentarily results in a rapid, uncontrolled chain reaction, releasing an immense amount of energy explosively [3].

# 3 Implementation

## 3.1 Model 1: $M_c$

This program captures the essence of the critical mass calculation. It provides a basic framework that can be further refined for more precise and complex simulations.

```
1  procedure cube(int x, int result)
2      result += x * x * x
3
4  procedure criticalMass(int Rc, int RcCubed, int density, int fourPi,
5                         int numerator, int criticalMass)
6      call cube(Rc, RcCubed)
7
8      numerator += (fourPi * RcCubed) * density
9      criticalMass += numerator / 3
10
11     numerator -= (fourPi * RcCubed) * density
12     uncall cube(Rc, RcCubed)
13
```

```
procedure main()
    int Rc
    int RcCubed
    int density
    int fourPi
    int numerator
    int criticalMass

    fourPi += 12

    // set initial values for calculation
    Rc += 3
    density += 2

    call criticalMass(Rc, RcCubed, density, fourPi, numerator, criticalMass
   )
```

In the `main` procedure we start by initializing the critical radius $Rc$, the density of the fissile material $Density$ and a rough approximation for $4\pi$. We then call the `criticalMass` procedure. In this procedure, we start by calling the `cube` procedure, to compute $R_c^3$. Then the numerator $4\pi R_c^3 \rho$ is computed and then divided by 3 to get the critical mass.

The value of $4\pi$ is approximated to 12 for simplicity. In a more precise implementation, a more accurate value should be used, and thus we would probably change units for the equation to still use integers. Example values are provided for $R_c$ and $\rho$. In a real-world application, these would be based on actual measurements or input values.

### 3.2 Model 2: $R_c$

This program provides a framework to calculate the critical radius in a reversible manner using Janus, ensuring that each computation step can be reversed.

```
procedure square(int x, int result)
    result += x * x

// from https://topps.diku.dk/pirc/janus-playground/#examples/sqrt
procedure sqrt(int num, int root)
    local int bit = 1
    from bit = 1 loop  // find exponential ball park
        call doublebit(bit)
    until (bit * bit) > num

    from (bit * bit) > num do
        uncall doublebit(bit)
        if ((root + bit) * (root + bit)) <= num then
            root += bit
        fi (root / bit) % 2 != 0
    until bit = 1
    delocal int bit = 1
    num -= root * root

procedure doublebit(int bit)
    local int z = bit
    bit += z
    delocal int z = bit / 2

procedure criticalRadius(int Rc, int Bg, int BgSquared, int pi,
                         int piSquared, int radius, int Rctemp)
    BgSquared += piSquared / (radius * radius)
```

```
28
29      call sqrt(BgSquared, Bg)
30
31      Rctemp += pi / Bg
32      Rc += Rctemp
33
34      Rctemp -= pi / Bg
35
36      uncall sqrt(BgSquared, Bg)
37
38      BgSquared -= piSquared / (radius * radius)
39
40  procedure main()
41      int Rc
42      int Bg
43      int BgSquared
44      int pi
45      int piSquared
46      int radius
47      int Rctemp
48
49      pi += 3
50      call square(pi, piSquared)
51
52      // set initial values for calculation
53      radius += 3
54
55      call criticalRadius(Rc, Bg, BgSquared, pi, piSquared, radius, Rctemp)
```

In the `main` procedure we start by initializing the variables $\pi$, $\pi^2$ and radius of the sphere. We then call the procedure `criticalRadius`, where we start by computing $B_g^2$ using the provided radius and $\pi^2$. We then call the `sqrt` procedure to compute $B_g$. In the end we compute the critical radius $R_c$ using the formula $R_c = \frac{\pi}{B_g}$.

The `sqrt` is taken from `https://topps.diku.dk/pirc/janus-playground/#examples/sqrt`.

The value of $\pi$ is approximated as 3 for simplicity. Example values for the radius is provided. In a real-world application, these would be based on actual measurements or inputs. This program would also benefit from using smaller units for a more precise answer.

### 3.3 Model 3: combination

This program integrates the previous models and allows for iterative adjustment of mass and density to determine the critical mass, ensuring that the system reaches criticality.

```
1  procedure cube(int x, int result)
2      result += x * x * x
3
4  procedure square(int x, int result)
5      result += x * x
6
7  // from https://topps.diku.dk/pirc/janus-playground/#examples/sqrt
8  procedure sqrt(int num, int root)
9      local int bit = 1
10     from bit = 1 loop  // find exponential ball park
11         call doublebit(bit)
12     until (bit * bit) > num
13
14     from (bit * bit) > num do
```

```
15          uncall doublebit(bit)
16          if ((root + bit) * (root + bit)) <= num then
17              root += bit
18          fi (root / bit) % 2 != 0
19      until bit = 1
20      delocal int bit = 1
21      num -= root * root
22
23  procedure doublebit(int bit)
24      local int z = bit
25      bit += z
26      delocal int z = bit / 2
27
28  procedure criticalRadius(int Rc, int Bg, int BgSquared, int pi,
29                           int piSquared, int radius, int Rctemp)
30      BgSquared += piSquared / (radius * radius)
31
32      call sqrt(BgSquared, Bg)
33
34      Rctemp += pi / Bg
35      Rc += Rctemp
36
37      Rctemp -= pi / Bg
38
39      uncall sqrt(BgSquared, Bg)
40
41      BgSquared -= piSquared / (radius * radius)
42
43  procedure criticalMass(int Rc, int RcCubed, int density, int fourPi,
44                         int numerator, int criticalMass)
45      call cube(Rc, RcCubed)
46
47      numerator += (fourPi * RcCubed) * density
48      criticalMass += numerator / 3
49
50      numerator -= (fourPi * RcCubed) * density
51      uncall cube(Rc, RcCubed)
52
53  procedure criticality(int mass, int deltaMass, int density,
54                        int deltaDensity, int Rc, int RcCubed, int fourPi,
55                        int numerator, int criticalMass, int pi,
56                        int piSquared, int radius, int Bg, int BgSquared,
57                        int Rctemp, int isCritical, int time, int maxTime)
58      // calculate critical Radius
59      call criticalRadius(Rc, Bg, BgSquared, pi, piSquared, radius, Rctemp)
60
61      from isCritical = 0 && time < maxTime loop
62           // calculate critical mass
63           call criticalMass(Rc, RcCubed, density, fourPi, numerator,
64                             criticalMass)
65
66           if mass >= criticalMass then
67              isCritical += 1
68           else
69              mass += deltaMass
70              density += deltaDensity
71           fi mass < criticalMass
72
73           time += 1
```

```
74      until isCritical != 1 || time >= maxTime
75
76  procedure main()
77      int mass                    // Current mass of the fissile material
78      int deltaMass               // Increment for mass in each step
79      int density                 // Density of the fissile material
80      int deltaDensity            // Increment for density in each step
81      int Rc                      // Critical radius
82      int RcCubed                 // Rc ** 3
83      int numerator               // (4 * pi * Rc ** 3) * Density
84      int criticalMass            // Critical mass
85      int pi                      // Approximate value of pi
86      int piSquared               // pi ** 2
87      int fourPi                  // 4 * pi (we use an approximation)
88      int radius                  // Radius of the sphere
89      int Bg                      // Geometric buckling
90      int BgSquared               // Bg ** 2
91      int Rctemp                  // Temporary variable for Rc calculation
92      int isCritical              // Indicator for achieving criticality
93      int time                    // Current time
94      int maxTime                 // Maximum simulation time
95
96      // set initial values of calculation
97      mass += 100                 // Starting mass of the fissile material
98      deltaMass += 4              // Define the mass increment per step
99      density += 2                // Starting density of the fissile material
100     deltaDensity += 0           // Define the density increment per step
101     pi += 3                     // Approximate value for pi (use a better
    approximation as needed)
102     fourPi += pi * 4            // Approximate value for 4 * pi
103     maxTime += 100              // Define the maximum simulation time
104     call square(pi, piSquared)
105     radius += 3
106
107
108     call criticality(mass, deltaMass, density, deltaDensity, Rc, RcCubed,
109                      fourPi, numerator, criticalMass, pi, piSquared,
110                      radius, Bg, BgSquared, Rctemp, isCritical, time,
111                      maxTime)
```

In the code above, we have developed a Janus program that models the criticality conditions of a fissile material by iteratively adjusting its mass and density. The program calculates critical mass and critical radius, ensuring that all operations are reversible, as required by the Janus programming language.

The program starts with two fundamental procedures: `cube` and `square`. The `cube` procedure calculates the cube of a given integer x and stores the result in result, while the `square` procedure calculates the square of a given integer x and stores the result in result. These operations are essential for the subsequent calculations of critical mass and critical radius.

The `sqrt` procedure calculates the square root of a given integer num and stores the result in root. It is taken from `https://topps.diku.dk/pirc/janus-playground/#examples/sqrt`.

The `criticalRadius` procedure calculates the critical radius Rc based on the geometric buckling Bg, the square of $\pi$, and the radius of the sphere. It first calculates BgSquared and then calls the sqrt procedure to get Bg. Finally, it computes Rc using the formula $\frac{\pi}{Bg}$.

The criticalMass procedure calculates the critical mass based on the critical radius Rc, its cube RcCubed, the density of the fissile material, and a constant approximation of $4\pi$. The critical mass is computed using the formula $\frac{4\pi R_c^3 \rho}{3}$.

The `criticality` procedure iteratively adjusts the mass and density of the fissile material to determine if criticality is achieved. It first calls the `criticalRadius` procedure to calculate the critical radius and then calls the `criticalMass` procedure to compute the critical mass. If the current mass is greater than or equal to the critical mass, criticality is marked as achieved. Otherwise, the mass and density are incremented, and the process repeats until the maximum simulation time is reached or criticality is achieved.

The main procedure initializes all the variables and sets their initial values. It defines the starting mass, density, increments for mass and density, and other constants such as $\pi$ and $4\pi$. The `criticality` procedure is then called to start the iterative process of adjusting mass and density to check for criticality.

By adjusting the variables, the model can be tailored to simulate different scenarios. The initial value of mass is set to 100. This represents the starting mass of the fissile material. To model different scenarios, this value can be adjusted. For instance, a higher starting mass might be used to simulate a more substantial initial amount of fissile material. The value of deltaMass is set to 4, which means the mass is incremented by 4 units in each iteration. This can be modified to simulate faster or slower increases in mass. A smaller increment value would model a more gradual increase, while a larger value would represent a more aggressive approach. The initial density is set to 2. This value represents the starting density of the fissile material. Different densities can be modeled by changing this initial value. For example, higher density values might be used to simulate denser materials. The deltaDensity is set to 0, meaning the density remains constant during the simulation. To model scenarios where density changes, this value can be adjusted. Increasing the density increment would simulate scenarios where the material becomes denser over time. The initial radius is set to 3. This value can be changed to simulate different geometric configurations of the fissile material. A larger radius might represent a more extensive material configuration, affecting the geometric buckling calculations. The value of $\pi$ is approximated as 3, and $4\pi$ is calculated accordingly. For more accurate simulations, better approximations of $\pi$ should be used. This would improve the precision of the critical mass and radius calculations. The simulation time is set to 100 iterations. This controls how long the simulation runs and can be adjusted based on the desired granularity of the simulation. Longer simulation times allow for more iterations, providing a finer resolution in determining criticality.

By adjusting these variables, the model can simulate a wide range of scenarios in nuclear physics, providing valuable insights into the behavior of fissile materials under different conditions. This flexibility makes the program a powerful tool for educational purposes, research, and practical applications in nuclear reactor design and safety analysis.

# 4 Discussion

The combined programs implemented here aim to model the conditions under which a fissile material reaches criticality by iteratively adjusting the mass and density. This comprehensive model integrates calculations for both critical mass and critical radius and provides a detailed simulation of neutron interactions and the physical properties required to sustain a nuclear chain reaction.

In nuclear physics, the behavior of neutron production and absorption is crucial for understanding the dynamics of a nuclear chain reaction. Neutron production occurs when a fissile nucleus, such as Uranium-235 or Plutonium-239, undergoes fission, releasing multiple neutrons. These neutrons can then induce further fission reactions, potentially leading to a chain reaction. Neutron absorption, on the other hand, involves neutrons being captured by nuclei, which may or may not lead to further fission. The balance between neutron production and absorption determines whether a chain reaction is self-sustaining, subcritical, or supercritical.

In our combined model, we did not use neutron production and absorption functions; however, it's worth discussing how these could be used as an initial step. Simplified models often represent neutron

production and absorption using basic modulus operations. While these simplifications allow for easy implementation and understanding, they do not accurately capture the complex physical interactions that occur in real nuclear materials. Simplified functions might assume linear relationships or fixed increments, which do not reflect the probabilistic and highly variable nature of neutron interactions.

Implementing more sophisticated models for neutron production and absorption would make the simulation more realistic. This could involve using probabilistic methods or detailed physical equations based on empirical data. Incorporating adaptive algorithms to adjust mass and density based on the current state of the system could provide a more accurate and efficient path to criticality. Allowing the simulation to run dynamically until criticality is reached or significantly beyond could provide more comprehensive insights into the behavior of the fissile material. Using real-world data for material properties and neutron interactions could greatly enhance the accuracy and applicability of the model.

Realistic modeling of neutron production and absorption should incorporate probabilistic methods. This involves using Monte Carlo simulations to account for the random nature of neutron interactions. Each neutron's path, likelihood of inducing fission, and chances of being absorbed would be simulated based on known physical probabilities.

Neutrons produced in fission have a spectrum of energies. Fast neutrons are more likely to escape the material, while thermal (slow) neutrons have a higher probability of causing further fission. Modeling the energy spectrum and interactions of neutrons at different energies provides a more accurate representation of the chain reaction dynamics.

Different materials have unique properties that affect neutron interactions. For instance, Uranium-235 has a different neutron absorption cross-section compared to Plutonium-239. Incorporating detailed material properties, such as cross-sections for absorption and fission, neutron scattering probabilities, and material density, can significantly enhance model accuracy.

The shape and configuration of the fissile material influence neutron behavior. Spherical geometries minimize neutron escape compared to cylindrical or irregular shapes. Modeling these geometric factors helps in understanding the spatial distribution of neutrons and their likelihood of escaping or inducing further fission [1].

By using these detailed modeling approaches, we can achieve a more accurate representation of neutron production and absorption dynamics. This results in more precise predictions of critical mass and behavior under various conditions. Probabilistic methods and energy spectrum considerations allow the model to realistically represent the stochastic nature of neutron interactions and differentiate between fast and thermal neutrons. Incorporating specific material properties ensures that the model accurately reflects the behavior of particular fissile materials, which is crucial for both reactor design and safety analysis. Geometric accuracy provides a realistic understanding of neutron distribution and escape probabilities, essential for designing efficient and safe nuclear reactors.

Changing the unit of measurement to a smaller scale can significantly enhance the precision of calculations in our model of neutron production and absorption, critical mass, and critical radius. This discussion explores how using smaller units can lead to more accurate simulations and a better understanding of the behavior of fissile materials.

Using smaller units reduces the impact of rounding errors. When calculations involve large values, rounding can introduce significant inaccuracies. By switching to smaller units, the magnitude of these errors diminishes, leading to more precise results. Smaller units allow for a finer resolution in measurements, meaning that changes in mass, density, and other parameters can be tracked more accurately. For instance, instead of increasing mass in steps of kilograms, we could use grams or even milligrams. This finer resolution enables more detailed tracking of incremental changes, resulting in a more nuanced understanding of the criticality conditions.

With smaller units, the model becomes more sensitive to minor changes in parameters. This heightened sensitivity is crucial for accurately capturing the threshold conditions for criticality. Small variations in mass or density that might be overlooked with larger units can be detected and accounted for, ensuring a more accurate representation of the physical processes.

In practical terms, adjusting mass and density in our current model involves changing the increments. Currently, mass is incremented in steps of 4 units and density in steps of 1 unit. By changing the unit of measurement, these increments can be made smaller, such as 0.004 units for mass and 0.001 units for density. This allows for a more gradual and precise adjustment of these parameters. Constants like $\pi$ and $4\pi$ should be represented with greater precision when using smaller units. For instance, using more decimal places for $\pi$ (e.g., 3.14159 instead of 3) can enhance the accuracy of calculations involving these constants. Similarly, calculations involving critical mass and radius should be adjusted to reflect the smaller units, ensuring consistency and precision throughout the model. Implementing a scaling factor can help transition from larger to smaller units. For example, if the original unit is kilograms, converting to grams involves multiplying by 1000. This scaling factor must be consistently applied across all calculations to maintain the integrity of the model[2].

Enhanced precision allows for a more accurate simulation of neutron interactions and the conditions leading to criticality. This is particularly important in nuclear physics, where small changes can have significant impacts on the behavior of fissile materials. More precise models provide better data for decision-making in nuclear reactor design, safety protocols, and research. Accurate simulations can help identify optimal configurations and safety margins, leading to improved efficiency and safety in nuclear operations. For educational and research purposes, having a high-precision model allows for a deeper understanding of the underlying physical principles. Students and researchers can explore the effects of small changes in parameters and gain insights that might be missed with less precise models.

Changing the unit of measurement to a smaller scale can significantly enhance the precision of calculations in our model. By reducing rounding errors, allowing for finer resolution, and improving sensitivity to parameter changes, the model becomes more accurate and reliable. This enhanced precision is crucial for accurately simulating criticality conditions and provides valuable insights for nuclear reactor design, safety analysis, and research. Implementing these changes ensures that our model remains a powerful tool for understanding the complex dynamics of fissile materials, ultimately leading to more informed decision-making and safer nuclear operations.

# 5 Conclusion

This study set out to model and simulate critical mass phenomena within the Janus playground, addressing three core problems: calculating the critical mass for various fissile materials, understanding the dynamics of a chain reaction, and determining the conditions for achieving and maintaining supercriticality. While we have made significant strides in adapting continuous mathematical models into discrete formats suitable for Janus and validating our results through controlled experiments, certain goals outlined in the abstract remain partially unmet.

First, while we successfully calculated the critical mass threshold and modeled the dynamics of a sustained chain reaction, the accuracy of these calculations was limited by the approximations used for constants like $\pi$ and the simplified representations of physical interactions. The model's reliance on integer inputs, while necessary for Janus, introduced a degree of imprecision that could be mitigated by implementing more sophisticated algorithms and using smaller units of measurement to enhance calculation precision.

Secondly, although we investigated the conditions necessary for achieving supercriticality, the study did not fully explore the detailed neutron production and absorption functions. Simplified models using basic modulus operations were discussed but not implemented, leaving a gap in accurately capturing the complex probabilistic nature of neutron interactions in real fissile materials. Future

work should incorporate these sophisticated models to improve the realism and applicability of the simulations.

Moreover, the challenge of maintaining consistency in forward and reverse simulations within Janus was not entirely resolved. While our models demonstrated some degree of reversibility, achieving full consistency remains an area for further development. The introduction of additional, more complex critical mass scenarios, as mentioned in the abstract, was not pursued in this study but should be considered in future research to fully test the capabilities of Janus.

In conclusion, while this project has successfully demonstrated the basic principles of critical mass and chain reactions within the constraints of Janus, further work is needed to refine these models for greater precision and realism. Enhancing the accuracy of constants, incorporating detailed neutron interaction models, and ensuring full reversibility in simulations will be crucial steps in advancing this research. The insights gained from this study lay a solid foundation for future explorations into more complex and realistic critical mass scenarios, ultimately contributing to a deeper understanding of nuclear physics within discrete simulation environments.

## References

[1] ScienceDirect. *Fundamentals of Nuclear Reactors*. Accessed: 2024-06-23. 2023. URL: https://www.sciencedirect.com/topics/engineering/neutron-production.

[2] Wikipedia contributors. *Arbitrary-precision arithmetic*. Accessed: 2024-06-23. 2023. URL: https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic.

[3] Wikipedia contributors. *Nuclear reactor physics*. Accessed: 2024-06-23. 2024. URL: https://en.wikipedia.org/wiki/Critical_mass.