

## Authors' Response to Referee Comment #2

on Manuscript "CLEO: The Fundamental Design for High Computational Performance of a New Superdroplet Model", now renamed "Cleo: The Fundamental Design of a New Superdroplet Model for High Computational Performance (v0.39.0)" by Clara J.A. Bayley et al. (2026)

Original Title	CLEO: The Fundamental Design for High Computational Performance of a New Superdroplet Model
New Title	Cleo: The Fundamental Design of a New Superdroplet Model for High Computational Performance (v0.39.0)
Author(s)	Clara J.A. Bayley et al.
MS No.	egosphere-2025-4398
MS type	Model description paper

We thank Referee #2 very much for taking the time to review our manuscript and suggest improvements, we are pleased with the level of detail they have gone into and we found their feedback both supportive and constructive for the manuscript. In the next section we address all of the general comments which the referee made; in the section thereafter we address the "other" comments.

Please note the updates to the dataset for this paper (<https://doi.org/10.17617/3.LNRKSJ>) are currently viewable from this link whilst they await publishing:  
<https://edmond.mpg.de/previewurl.xhtml?token=3eecde11-6c96-473f-9037-bfb58344d59b>.

### Response to General Comments

We thank the Referee for the general comments they made on the manuscript. Here we address each of them individually and to aid readability we have re-arranged them in chronological order.

#### Title

- *The title is missing software version, which is required by GMD guidelines. Moreover, the title suggests that a new model is introduced, but the paper outlines a new implementation of a "classic" model.*

We have renamed the title to include the version of Cleo presented in this paper, it is now "[Cleo: The Fundamental Design of a New Superdroplet Model for High Computational Performance \(v0.39.0\)](#)". In the second point below regarding the introduction, we also explain our decision to remove the acronym for "CLEO" and name the model simply as "Cleo". In the title we have chosen to describe Cleo as a "superdroplet model" rather than a "superdroplet method model" to make sure it is readable, however, in the main text we make clear Cleo is a new model which uses the (not-new) superdroplet method, e.g. (P3 L79): "[Hence we are creating Cleo: a novel implementation of SDM \[...\]](#)", and we also now more precisely define "SDM" as the superdroplet method (see first comment under "General" below, regarding the SDM acronym).

#### Abstract and Introduction

- Here we address the two points regarding the abstract and introduction together, these were:
  - *Starting with the abstract, there are several bold statements regarding optimal performance, efficient access patterns, cache efficiency, load balancing, and speed-ups, which later on in the text are clarified as based on intuition rather than comparison against alternative implementations. Matching the abstract and introduction with the content of the paper will improve reception.*
  - *The abstract focuses on "marketing", is repetitive, and does not convey a summary of what is in the paper (e.g., what kind of simulations were performed); suggest rewriting focusing on the key results from the presented development, and avoiding vague phrasing (e.g., a diverse range of computer architectures, a high degree of flexibility, ready to be used for understanding warm-cloud processes; all these would best be changed from "a"*

to "the" statements: which architectures, what kind of flexibility, which processes and what do you mean by understanding?)

We thank the referee for pointing out the Abstract did not well convey the content of the paper and the wording was often too vague and/or bold. We've completely re-written the abstract in order to take on board these criticisms, and we think the Abstract now more correctly summarises the paper and its wording is much more precise. Also to better match with the content of the paper, we've softened the wording in P3 L94- P4 L99 of the introduction: "we chose Cleo's memory layout **with the aim of** economically allocating memory and optimising memory access patterns for loops over superdroplets, whilst compromising on other aspects of performance, [...]. Secondly, we have **sought to** maximise Cleo's freedom to allocate resources on massively parallel heterogeneous computer architectures; [...], **in doing so we aim to be able to allocate resources efficiently.**".

- *Furthermore, I suggest to make it clear upfront in the abstract and introduction to the paper that the development - in the stage presented in the paper - is not tested at exascale, not even in a multi-node setup, and only with one-way coupling with a prescribed-flow driver*

We thank the referee for encouraging us to better align the expectations set by the introduction and abstract with the content of the manuscript. We think the acronym we gave Cleo was one of the key reasons the expectations did not align and upon further consideration we also find this acronym not only unhelpful in this context but also restrictive for Cleo's future development (we may expand Cleo beyond cloud microphysics and/or use Cleo on non-exascale systems). We've therefore reverted to using Cleo as a proper noun, as it was originally intended to be, in tribute to the real-life "super-women", Cleopatra and the anonymous mathematician whose pseudonym was Cleo. The first paragraph introducing Cleo now reads (P3 L79-81) "Hence we are creating Cleo: a novel implementation of SDM aiming to model warm-cloud microphysics efficiently on top-tier and exascale high-performance computers. Cleo's name is a tribute to two "super-women", Cleopatra VII Thea Philopator and the pseudonymous mathematician Cleo."

Additionally, we have made several amendments to the paragraph thereafter (P3 L82- P4 L99) to make it very explicit what this paper does and does not contain, most notably the second sentence of the second paragraph states "Specifically, this paper explains how we exploit the massively parallel heterogeneous resources and the hierarchical memory layout of individual nodes of high-performance computers.", and P3 L89-92 "In a follow-up paper we will therefore present Cleo's distributed-memory parallelism (MPI domain decomposition) and evaluate its performance specifically for LES **across multiple nodes of** top-tier and exascale high-performance computers. With regard to Cleo's fundamental computational structure **on single nodes**, [...]" . We've also amended P20 L518-521 along similar lines to state: "As such, we will evaluate Cleo's distributed-memory parallelism in an follow-up paper specific to LES conducted across multiple nodes of top-tier and exascale high-performance computers. This paper meanwhile discusses Cleo's fundamental computational design and it's performance on shared-memory architectures (single nodes)."

## Edmond dataset(s)

- *the Edmond dataset metadata keywords misleadingly include: Climate Modelling, LES, Exascale Computing, ICON;*  
We thank the reviewer for pointing this out and we agree it is indeed misleading. We originally chose to include those keywords because our current work and future publications is/will be relevant to those keywords; however we realise that does not benefit this dataset and this manuscript's fundamental model description. We've therefore replaced the keywords with "Superdroplet Model, SDM, Cloud Microphysics Modelling, HPC" and "Cleo".

## Kokkos Thread Parallelism

- *mention alternative parallelisation strategies for the SDM (e.g., where Monte-Carlo collisional growth parallelisation is done in parallel across all pairs in multiple cells of a given subdomain, rather than - IIUC - serially within cell);*

The parallelisation is done over superdroplets, over grid-boxes, and over superdroplets in grid-boxes hierarchically whenever desired. The strategy for parallelism is flexibly determined by Kokkos depending on the architecture, or manually by the user. We've now made this clearer with additional text in P9 L230-234: "When loops over superdroplets are nested inside loops over grid-boxes, as occurs during microphysics and superdroplet motion but not necessarily during initialisation or data output, hierarchical parallelism is invoked;

that is both the outer loop over grid boxes and the inner loop over superdroplets are parallelised. The exception is if Kokkos (or manually the user) determines that for the given hardware available the number of grid-boxes or superdroplets is too small for it to be worthwhile to parallelise the loop.”

## MPI Domain Decomposition

- *elaborate on the technological challenges stemming from the spatially uneven resource requirements of particle-based simulation (i.e., grid cells with clouds vs. grid cells without clouds), and from the fact that the employed simulations inherently cover aerosol budget (implementing aerosol sources should also benefit from the proposed memory layout);*

Based on this feedback, we've greatly elaborated on our discussion of MPI domain decomposition in Section 3.2 (P9 L239- P10 L269), especially in lines P10 L248-262 where we discuss specifically the technological challenges of particle-based, in our case superdroplet, simulations with regard to load balancing across computational resources. In P10 L254-58 we include specific mention that the choice of boundary conditions, including aerosol sources as well as lateral sinks of superdroplets, are one source of challenge because they can unbalance workloads: “The major challenge with such superdroplet-based domain decompositions is that superdroplets (and cloudy regions) move around the domain and superdroplets may be added or removed depending on a simulation's boundary conditions (e.g. its aerosol/superdroplet sources and lateral superdroplet sinks). The consequent changes in the number and activity of superdroplets changes the workload of MPI processes and a better optimised domain decomposition would dynamically adapt to restore balance.”.

## Monoids

- *mention alternative technologies as how they compare to the employed C++20;*

We presume this point mainly refers to the section describing our computational monoids since the other infrastructures/software we mention (arrays, classes/structures, MPI, YAC and Kokkos) are not C++ specific. We've now re-structured the section which introduces monoids (see next bullet point) and included a new paragraph (P12 L327-333) discussing what other technologies would also be suitable as opposed to C++20. This paragraph reads: “There are many other plausible ways to express monoids in code. Within older C++ standards (pre-2020) one could instead use for example a base class, a trait, or even just constraints written on a piece of paper. However, base classes would create more restricted, closed definitions of monoids, and the latter options would be more verbose and error-prone. Based on how we defined computational monoids above, other compiled or just-in-time compiled languages could also implement monoids computationally with the same overall behaviour as our monoids in C++. Naturally, languages with object-oriented programming paradigms are better-suited to implementing monoids since monoids are fundamentally described by set theory.”.

- *Given that the paper's intended audience seems to be tech-aware research software engineering community, I suggest to redact the technical parts of the paper with references to alternative technologies. For instance, the “flexibility” featured in the title of Section 4, would likely be termed “workarounding the stiffness of C++” from a just-in-time compiled language perspective. Covering it in the paper is fine, but I suggest rephrasing - making sure that the paper is readable, approachable and appreciable for audiences ranging from (modern) FORTRAN to Julia coders. For instance, when writing about C++20 concepts, it would be apt to refer to Julia multiple dispatch alternatives.*

Also here we presume that “technical parts” is primarily referencing to our section on monoids because we are reasonably confident that the other technical parts (arrays, classes/structures, MPI, YAC and Kokkos) are already familiar with our “tech-aware research software engineering community”, particularly climate model and LES developers, and that they are understandable to (modern) FORTRAN and Julia coders even if they do not know the details of YAC or Kokkos. If this is not the case, we are happy to be pointed to any other specific technical parts of the manuscript which are not yet satisfactory so that we can improve them.

To address this referee's point in regard to monoids, we've restructured the section introducing monoids and we have generalised the first mention of computational monoids therein to be agnostic of C++ so that it appeals to a wider audience (P11 L297-304): “We apply the idea of monoids computationally by creating objects in code that can be combined pairwise and associatively with one-another. To do this, first we define a set of constraints a code-object must satisfy in order to be allowed to be combined with another such object. In other words, we define constraints an object must satisfy to be part of a certain code-based monoid set. Secondly, alongside these constraints, we define a function (an associative binary operation) which determines exactly how two objects which obey these constraints are combined together such that the return of the function is a new object which itself satisfies the given constraints. In this way, any number

and permutation of such objects can be combined sequentially to create the final object that is used at run-time.”. In combination with the rest of the section, we think this also makes clear that the section is truly about flexibility through monoids, in any language but in our case C++, and not about working around C++ stiffness. Later on we then describe our specific C++ implementation with template-metaprogramming and C++20 concepts (P12 L321-326): “To implement monoids in Cleo, we take advantage of template meta-programming with C++20 concepts. Specifically, for a given monoid set we use a C++20 concept to impose the constraints of the set on templated types, and we overload the right-shift operator (`operator>>`) to implement the function for the associative binary operation. We then create structures/-classes which obey the C++ concept and can thus be combined during compilation, in whatever way desired, to instantiate the templated type that is used throughout the code at run-time. Using C++20 concepts is not essential, but has the added benefit of more comprehensible error messages than ordinary template meta-programming in C++.” and following this we include the new paragraph on alternative technologies as mentioned in the bullet point above.

We have purposefully not generalised the subsequent sub-sections (Section 4.1 and Section 4.2) which describe in detail the monoids Cleo defines for microphysical processes and observers. This is because those monoid definitions are specific to Cleo and the purpose of those sections is not to define general monoids for microphysics and data output. The purpose of those sections is rather to explain the specific implementations of the monoids that are essential for understanding how to use and develop Cleo, as is of course necessary to adequately describe Cleo in its model description paper. We have modified P13 L344-L348 to be more clear that the following sections are Cleo specific: “Thus, to ensure run-time performance whilst still enabling maintainability and flexibility, Cleo defines monoids for microphysics and data output. Section 4.1 details our monoid for microphysical processes, namely the C++ concept which defines microphysical processes, the structure which acts as the identity element of the set, and the right-shift operator used to combine microphysical processes with one another. Likewise, Section 4.2 details the monoid set for “observers”, the elements of which can be used for data output.”

- *elaborate on the implications of the implementation choices in terms of maintenance (e.g., the Monoids should facilitate unit testing, although the contents of the ‘tests’ folder in CLEO does not seem to leverage it yet...).*

We added on P12 L339-343: “[...] branching would add computational expense and reduce the code's modularity. Modular as opposed to monolithic code is more readable and easier to maintain because it can be understood in piecewise units and each unit can be tested and developed independently, as well as concurrently. Although at the time of writing Cleo does not yet have unit tests for its monoids, they are still modular units of code and thus facilitate code readability and maintainability.”. You're quite right to point out that we haven't yet found the time to implement unit tests in Cleo, we currently use the examples as integrated test cases instead (see the examples folder and our documentation for details: <https://yoctoyotta1024.github.io/CLEO/usage/examples/examples.html>). We realise this is by no means ideal, nor best practise, and we do intend to write unit tests in the near future. Since we don't yet have them, it would be inappropriate to be specific about our monoids facilitating unit testing in our code yet; nevertheless they do facilitate making Cleo maintainable and readable in the sense they make Cleo modular and hence easy to read and edit/develop.

## General

- *The SDM acronym was introduced by Shima et al. as Super-Droplet Method; here it is deciphered as Super-droplet Model (not Method) - is it intentional? Also, it seems unclear what the authors mean by SDM. On page 1, lines 31-32, SDM is used in a broad sense of super-particle microphysics with works employing contrasting particle-collision representations cited (probabilistic, deterministic, super-particle-conserving and not), and on page 17, lines 418, SDM is used to label three works not modelling particle collisions at all; on page 1, line 42, SDM is associated with “its coalescence algorithm” implying that SDM is used in the specific sense of Shima et al. 2009 Monte-Carlo algorithm. Yet, on page 6, line 146, there is a statement about “CLEO's SDM algorithms” (plural). This is in many ways misleading. Please state at the beginning, what is meant by SDM: particle-based microphysics methods (which predate SDM, and would likely better be called particle-based microphysics), Shima et al. Monte-Carlo algorithm; the whole set of schemes from the SDM paper, or something else? This is also important given the later conclusions about random shuffling bottlenecks - the paper does not provide enough context to understand where, when and why the shuffling is needed.*

Thank you for pointing this out, we had not thought carefully enough about how we define SDM. We've now corrected the first mention of the acronym (P2 L36) to be “The Super-Droplet Method (SDM; Shima et al., 2009), [...]” and clarified how we use the term “SDM” (P2 L42-48): “Several SDM implementations already exist, (e.g. Shima et al., 2009; Arabas et al., 2015; Naumann and Seifert, 2015; Brdar and Seifert,

2018; Jaruga and Pawlowska, 2018; Shima et al., 2020; Chandrakar et al., 2021; Bartman et al., 2022; de Jong et al., 2023; Matsushima et al., 2023) and are similar to other particle-in-cell-based cloud microphysics models (e.g. Sölch and Kärcher, 2010; Andrejczuk et al., 2010; Riechelmann et al., 2012; Hoffmann et al., 2015), the key difference being SDM's profound convergence property. As the number of superdroplets increases, the model, including its algorithm for droplet collision-coalescence, tends towards what we expect from explicit individual particle simulations (Shima et al., 2009). ". We thus clarify that by SDM we mean the Lagrangian particle-in-cell-based microphysics with the collision-coalescence algorithm as per Shima et al. 2009. Correspondingly we've updated P20 L509-510 to point to papers which use superdroplets according to that paper "[.] for example in studying the role of turbulence during droplet growth (Li et al., 2018, 2020; Grabowski and Thomas, 2021; Chandrakar et al., 2024; La et al., 2025)." (the incorrect citations where human-error citing papers of the same first author but incorrect year, thank you for spotting this!).

We've also clarified in P6 L164-165: "the algorithms Cleo uses to conduct SDM microphysics" and P19 L463-464 we've clarified that the shuffling is required in order to enact SDM collision-coalescence: "This is because the bottleneck of microphysics is the random shuffling of superdroplets required by the method for SDM collisions".

- *Figures 1, 2, 3, 6, 7, 8 and 9 are supplied in raster graphics format, please replace all figures with vector graphics suitable for publication. Replotting the figures, ensure that font size roughly matches the font size in the main body of the paper text - currently most labels in the plots are too small.*

We have re-made all the figures as .pdf files, and increased all the font-sizes.

## Response to Other Comments

We thank the Referee for the other comments they made on the manuscript. We've addressed each of these individually with the changes shown in the tracked changes document. For some comments, we provide additional responses in the following:

- *abstract: "conservative memory usage" is mentioned; however, first: it is not explained in the paper how an alternative memory usage would imply more memory needs (in contrast, it is mentioned that the employed sorting algorithm doubles the memory footprint!); second: the paper states that CLEO abstracts away such details as if the storage layer is based on linked lists or contiguous arrays.*

This comment no longer applies to our re-written abstract. Nevertheless, we would like to clarify here why we used the term "conservative memory usage" to describe Cleo originally. We used the term because (P6 L166-169; now it's own paragraph to make more clear) "Single, contiguous arrays of grid-boxes and superdroplets efficiently handle memory allocations. As opposed to having superdroplets scattered in memory, e.g. by having a separate array of superdroplets for each grid-box, having a single array of superdroplets requires less frequent memory (de)allocations when superdroplets move around the domain and therefore also avoids the need for buffers which would increase the memory consumption.". Having said that, it's unclear if this saves memory overall because, as rightly pointed out, the current sorting algorithm we use doubles the superdroplet memory footprint.

The second part of this comment regarding Cleo's storage layer also no longer applies to the rewritten abstract. Nevertheless, to resolve any confusion the previous abstract may have caused we would like to respond here. Cleo uses contiguous arrays (specifically Kokkos::View and Kokkos::DualView instantiations, see <https://kokkos.org/kokkos-core-wiki/API/core/view/view.html>). Their layout (right, left, stride, or even custom defined) is indeed abstracted, but the allocation is not. We state on P5 L157-158 "the structures for each grid-box are stored together in a single array (an Array of Structures; AoS), and likewise superdroplets are stored in their own AoS, in a contiguous block of memory completely separate from grid-boxes." and then on P9 L225-228 we state "[Kokkos determines] how to appropriately allocate the memory and execution resources to achieve high performance. For example, it determines whether to use column- or row-major layouts for arrays of grid-boxes and superdroplets in order to avoid cache misses, and it chooses task sizes for threads of parallelised loops which optimise cache blocking.". We mention linked lists only in the discussion of how grid-boxes "view" the superdroplets associated with them at any given time as an alternative to our current method using pointers (strictly speaking a Kokkos::Subview): (P5 L132-134) "[The view] is an indicator to their location(s) elsewhere in memory. For example, the view could be a linked-list of pointers to individual superdroplets, or it could be two pointers to the start and end of a sub-section of a larger array of superdroplets (as it is in our current implementation).".

- There were several comments regarding the first paragraph of the manuscript on P1. These were:

- page 1 / line 13: "observations of warm-rain" is a very broad statement; this very first sentence of the paper would benefit from clarification on what kind of observations you have in mind, what is the type of disagreement (total amount, timing, size spectrum?). For instance, the abstract of the vanZanten work cited to support the disagreement statement states: "The ensemble average of the simulations plausibly reproduces many features of the observed clouds..."
- page 1 / line 16: "obscurity in cloud microphysics" calls for elaboration; obscurity suggests a lack of process understanding (next sentence also mentions "fundamental gaps in our microphysics knowledge"). It is fair to say we don't know everything about cloud microphysical processes, but especially given that the study deals with warm-rain clouds, it would be best to explain what is unknown and how much are the knowledge gaps limiting us vs. how complex are the challenges in modelling well understood but multi-scale and non-linear processes.
- page 1 / first paragraph: overall, I find the first three sentences (with 15 references) "obscuring" the idea of the paper, rather than introducing it. Clearly subjective, but perhaps the two above points can help in rephrasing it.

We've clarified we mean macrophysical observations of warm-rain, i.e. its spatial and temporal pattern (P1 L15-17) "Persistent and large discrepancies between observations of the spatial and temporal patterns of warm-rain and Large Eddy Simulations (LESs) have been ascribed to cloud microphysics for decades (e.g. Randall et al., 2003; Abel and Shipway, 2007; Ackerman et al., 2009; vanZanten et al., 2011) (see also summaries in Khain et al., 2015; Morrison et al., 2020)." In the sentence thereafter we've removed the words "obscurity in" so the sentence states (P1 L18-19): "With the advent of Global Storm Resolving Models (GSRMs) which remove the need to parametrise convection, ~~obscurity in~~ cloud microphysics is now one of the leading sources of uncertainty in global models too".

The aim of this paragraph was to emphasise there are two distinct sources of uncertainty in conventional, Eulerian cloud microphysics schemes and these stem from knowledge and model uncertainty, but we realise we hadn't managed to make this point properly. To make it better we've added the sentences (P1 L20- P2 L25) "Much of the uncertainty in both LES and GSRMs arises from epistemic uncertainty, that is uncertainty caused by our insufficient knowledge of microphysical processes and condensate attributes. However, distinct from that, these models also suffer from uncertainty caused by the Eulerian microphysics schemes we conventionally use to represent the knowledge we already have (Morrison et al., 2020)." This also points out these are Eulerian models to contrast with the Lagrangian superdroplet method later on.

- page 1 / line 21: super-particle model is later on labelled as Lagrangian, perhaps labeling the "conventional" methods as Eulerian could help in pointing out the difference?

Thank you for the suggestion, we have included it in the sentence already mentioned above (P1 L23-24) "However, distinct from that, these models also suffer from uncertainty caused by the **Eulerian** microphysics schemes we conventionally use [...]" and in P2 L26: "Conventional, **Eulerian** bulk/moment and bin microphysics schemes have intrinsic deficiencies, [...]"

- page 1 / line 43: the commonly used term is "embarrassingly" rather than "extremely parallelisable"; given the scope of the paper, it seems reasonable to point here out that the reason for it is the non-overlapping pair sampling, while the GPUs enable one to leverage it.

We've corrected "extremely" to "embarrassingly" and added more explanation on P2 L52-54: "Moreover SDM is embarrassingly parallelisable because its algorithms for microphysics and motion act independently on individual superdroplets or non-overlapping pairs of superdroplets, [...]"

- page 1 / line 46: this statement applies to bin models only, while "conventional" has been associated with both bulk and bin earlier; moreover it seems worth mentioning that for "bin" models, two attributes are already a rarity (Lebo & Seinfeld 2011), while employing more attributes is needed to resolve mixed-phase processes or chemical ageing in the context of aerosol-cloud interactions.

We've amended "conventional" to "bin" in the statement (P2 L56-60): "Its scaling with increasing microphysical complexity is also better than that of **bin** models once the number of superdroplet attributes is greater than about four (Shima et al., 2009).", and also added that "This makes SDM better suited to modelling droplets with more complexity, which is rarely done in bin and bulk schemes and is necessary to better represent important microphysical processes, for example riming in mixed-phase clouds (Brdar and Seifert, 2018) and aerosol chemistry in aerosol-cloud interactions (Jaruga and Pawlowska, 2018)."

- page 4 / line 118: suggest splitting SCALE and ICON references into separate parentheses;

Done (P5 L136-137), the text now reads "This is because LES grids can vary substantially, for example SCALE uses a Cartesian Arakawa-C grid (Sato et al., 2015; Nishizawa et al., 2015), whereas ICON uses a icosahedral-triangular Arakawa-C grid (Hohenegger et al., 2023), [...]"

- page 6 / line 153: “sequentially” suggests serial execution

This is certainly not what we meant (quite the opposite!) and upon reviewing this sentence we realise it lacked a proper explanation of the benefits to memory access patterns of our memory layout. We’ve therefore added considerable explanation on P7 L170-185, stating “One advantage is that we maximise locality of reference, [...] The second major advantage [is that our memory layout causes] less contention in memory access, [...] In many respects our memory layout amounts to cache blocking, [...]”. This re-written paragraph also removes the misleading use of the word “sequentially”.

- page 7 / line 170: I do not agree with the statement “SoA layout would need to perform sorting/shuffling on the array for each individual sub-component of the superdroplets separately”, since an SoA layout could feature an indirection layer mapping particle id through a (single) permutation vector;

We’ve clarified we meant “any SoA layout which still maintains the advantages for memory access patterns of ordered superdroplet data” (P8 L196-199).

- page 7 / line 189: “to advect thermodynamics” sounds too jargonic (and suggests that it is CLEO which does the advection, but she is not IIUC);

We’ve clarified what we mean on P8 217-218 “Whereas Cleo advects superdroplets according to the fluid flow (wind fields) itself, to advect thermodynamic fields (temperature, pressure etc.) Cleo must be coupled to a host dynamical driver capable of advection” as well as in P11 L278-279 “Whilst Cleo can thus advect superdroplets, to advect thermodynamic fields as per the fluid flow, Cleo must be coupled to a dynamics-solver with a fluid-dynamical core.”.

- page 16 / line 363: please elaborate on how a “2-D kinematic flow” was adapted to a 3-D domain?

We’ve clarified on P18 L452-453: “To extend the simulation into 3-D, we replicate the 2-D (x-z) flow uniformly along the additional (y) dimension. ”

- page 16 / line 364: feedback from microphysics is a different thing than relaxation - likely “rather than” is a wrong phrase here;

“rather than” has been corrected with “nor” (P18 L454): “[...] without feedback from microphysics **nor relaxation** towards the initial profiles.”.

- page 16 / line 381: suggest rephrasing “suffers from microphysics” and “suffers from motion”;

we rephrased “suffers from” to “is/are limited by” (P19 L471-472): “CUDA speed-up is limited by microphysics (Figure 8b), while OpenMP and C++Threads speed-ups are limited by motion (Figure 8c).”.

- Here we respond to the two comments regarding serial vs parallel random shuffling and random number generators together. The comments were:

- page 17 / line 374: if only serial random shuffling is featured, it is worth highlighting earlier on;
- since serial shuffling of particles has been identified as a bottleneck, parallel alternatives for permutations (e.g., MergeShuffle, arXiv:1508.03167) or parallel pseudorandom number generation (e.g., cuRAND for CUDA) should be considered... but on page 17, line 406 “creation of thread-safe random number generators” is mentioned - unclear;

Shuffling is inherently expensive, even when parallelised, and we mention this earlier in the manuscript on P3 L91-93: “With regard to Cleo’s fundamental computational structure on single nodes, the memory and particle transport expenses are prevalent when accessing, sorting, and shuffling superdroplets.” and in the discussion of Cleo’s memory layout we state P8 L196-197: “the speed of algorithms which sort/shuffle superdroplets during motion/microphysics scale only with the number of superdroplets.”. With regard to considering parallel alternatives for permutations, we have now stated more clearly that a different algorithm may improve performance on P19 L474-477: “The bottleneck is again the shuffling algorithm we use during collision-coalescence, presumably due to the relatively slow clock-speed of a single GPU thread during the serial step of the Fisher-Yates algorithm we use for shuffling. A different (possibly parallelised) algorithm may therefore improve performance,” and P21 L549-550 “Using a different shuffling algorithm may be more optimal for GPUs, [...]”, but we have now removed the first mentioning of the shuffling algorithm being serial in P19 L464 “[...] the bottleneck of microphysics is the (**serial**) random shuffling of superdroplets [...]” and we have softened the wording around suggesting it be parallelised (“(possibly parallelised)”) because we feel the emphasis on a parallel as opposed to serial shuffling algorithm here is misleading.

Considering this further, the MergeShuffle parallel random shuffling algorithm does not offer any significant speed-up over the Fisher-Yates serial shuffling algorithm for problem sizes less than  $10^7$  (arXiv:1508.03167

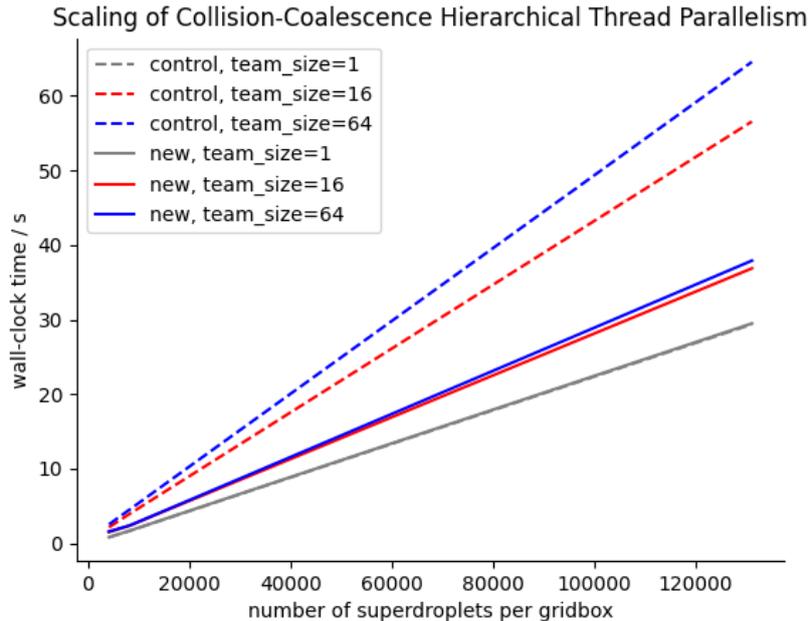


Figure 1: time for shuffling in a box-model performance test of the serial Fisher-Yates algorithm (“control”; dashed) and MergeShuffle (“new”; solid) for different numbers of CPU threads (“team\_size” = 1, 16 or 64). In the case of Fisher-Yates with multiple threads, a single thread performs the shuffling and then synchronises with the other threads afterwards. The single-thread, i.e. serial, implementations of Fisher-Yates and MergeShuffle show the best performance here.

Fig. 1). We shuffle the superdroplets for each grid-box separately to enact collision-coalescence in each, distinct, grid-box (i.e. volume) and it would be *extremely* extraordinary to have  $10^7$  superdroplets per grid-box — we usually have no more than  $\mathcal{O}(100)$ . We therefore wouldn’t expect to see any improvement by using this parallel random shuffling algorithm. Nevertheless, we tested the MergeShuffle algorithm in a box model but indeed we found the parallelised MergeShuffle algorithm does not offer any significant speed-up over the standard Fisher-Yates serial algorithm (Figure 1); moreover we found that with multiple threads, albeit CPU ones, it even performs slower than the single-thread tests. We choose not to mention this in the manuscript, mainly because it would distract from the focus of analysing Cleo’s performance, and is not necessary for the conclusions of the performance analysis, most importantly that whilst Cleo’s shuffling algorithm is the bottleneck of GPU builds on a single node, it is unlikely to be the bottleneck of larger, more complex simulations and so before conducting any further optimisation of the shuffling we should first assess Cleo’s performance with the transport of superdroplets between nodes, additional microphysics, and/or two-way communication with a dynamics-solver with a fluid-dynamical core.

With regard to considering parallel pseudorandom number generation: our thread-safe random number generators do indeed generate pseudorandom numbers in parallel and we have added the word “parallel” in their description to clarify this on P20 L497 and P20 L499: “thread-safe **parallel** random number”. (Note as per the Kokkos documentation: “in contrast to CuRAND, none of the functions of the pool (or the generator) are collectives, i.e. all functions can be called inside conditionals.”; <https://kokkos.org/kokkos-core-wiki/API/algorithms/Random-Number.html>).

- page 17 / line 411: “can divided” missing “be”; Corrected on P20 L502.
- page 19 / lines 472-475: the “Code availability” and “Code and data availability” sections should be merged, and the licensing terms of the code and its key dependencies should be stated.

We have merged the two sections and re-written them following the guidelines from the GMD data policy ([https://www.geoscientific-model-development.net/policies/code\\_and\\_data\\_policy.html](https://www.geoscientific-model-development.net/policies/code_and_data_policy.html)). The core dependencies of Cleo v0.39.0 are: a GCC or Intel compiler for C++20, CMake, Kokkos, and yaml-cpp, which is also stated in Cleo’s documentation and now in the dataset too. The full code and data availability section now reads “The current version of Cleo is available from its GitHub page: <https://github.com/yoctoyotta1024/CLEO> under BSD 3-Clause license, alongside its documentation: <https://yoctoyotta1024.github.io/CLEO>. Version v0.39.0 is described and tested in this paper and all the code, including this Cleo version, as well as the results

included this paper are archived in the dataset on Edmond under DOI <https://doi.org/10.17617/3.LNRKSJ> (Bayley, 2025). The core dependencies of Cleo v0.39.0 are: a GCC or Intel compiler for C++20, CMake, Kokkos, and yaml-cpp.”.

- In response to the comments regarding the formatting and references which were:
  - *in references, 19 entries have malformed URLs: <https://doi.org/https://doi.org/...>*
  - *in references, there are discussion-stage papers cited, for which accepted peer-reviewed papers are available: Yin et al. 2023, Matsushima et al. 2023;*
  - *in references: please double check bibliography formatting, e.g., the Takasuka et al. 2024 paper has its 2023MS003701 id given four times*

We have fixed the formatting of the references and updated the entries for discussion-stage papers that are now accepted peer-reviewed papers.

## Additional Revisions

As well as taking on the feedback of the Referee, we have made some additional minor changes to the original manuscript listed here:

- We have re-phrased the cost estimate in P21 L556- P22 L564 to compare with ICON’s strong-scaling and to assume that cost overhead could also come from communication with the dynamics-solver, not just particle transport via MPI. We also consider the “extremely conservative estimate” to be a factor of 100 slower rather than 1000. The cost estimate now reads: “For a conservative estimate on the cost, let us assume that the overhead from particle transport via MPI and communication with the dynamics-solver increases Cleo’s wall-clock by a factor of 10. Then, by using 1000 NVIDIA A100 GPUs (250 “Levante-like” GPU-nodes), the SDM setup from the performance tests presented here suggest that  $\mathcal{O}(10^7)$  superdroplets per MPI process would need approximately one second of wall-clock time for every second of simulated time and have an order of magnitude less memory consumption than the maximum on each node. That would make Cleo’s throughput approximately a factor of 10 less than the strong-scaling limit of ICON (Klocke et al., 2025), and even for an extremely conservative estimate, where Cleo’s performance is slower by a factor of 100, a simulation of several hours that takes several weeks to run is still conceivable.”.
- Added more references for largest superdroplet simulations known to this author on P3 L73. Now: “(Sato et al., 2017; Chandrakar et al., 2021; Matsushima et al., 2023; Yin et al., 2024)” instead of “(Sato et al., 2018; Shima et al., 2020; Matsushima et al., 2023)”.
- P20 L525 added “try to minimise cache loading and contention”.