We would like to thank the reviewer for their constructive assessment of our submission. We will respond to each point in the order listed by the reviewer. We have highlighted the reviewer's comments in yellow and the changes made to the manuscript in response in blue.

(Technical report) The quantitative results are compelling and offer valuable insights for optimizing workflow submissions on congested HPC systems. However, there are still significant issues. I agree with another reviewer that the submission is much like a technical report, instead of a scientific paper. Further more, several technical problems should be addressed.

We appreciate the feedback and acknowledge that our submission appears to be a technical report. To address this, we revised the manuscript to provide more contextualization of our work within the current state of research and clarified the hypothesis.

Specifically, we expanded the introduction of this work to include a thorough state-of-the-art overview. The field of workflows is notoriously compartmentalized, but we have covered examples from multiple fields. Among the changes, we have included references to tools used in life and material sciences. Additionally, we have included pilot-job systems whose community will also benefit from this work because they also aggregate tasks into a large submissions. These changes were included in lines 53–59 and read as follows:

Aggregation was implemented in other fields, with different degrees of sophistication. In Earth Sciences, Mickelson et al. 2020 suggested using Cylc's feature to submit multiple jobs to reduce the queue time. Both Aiida (Huber et al. 2020) and Snakemake (Mölder et al. 2021)— from the material and life sciences fields, respectively — provide a way of to submit multiple workflow tasks as a single job. The former implements this via a ``metascheduler'' plugin (HyperQueue plugin), while the latter refers to aggregation as ``grouping.''

Moreover, Pilot-job systems were developed to increase workflow throughput, but later evolved into more sophisticated solutions (Turilli et al. 2018). These systems are characterized by implementing "resource placeholders, multi-level scheduling, and coordination patterns to enable task-level distribution and parallelism on multi-tenant resources." One major example of a modern pilot-job system is Radical-PILOT (Merzky et al., 2021)

Wrappers share some of the features of a typical pilot-job system. Wrappers provide a simpler ``resource placeholder'', where all the task requests are added into a large submission. However, their objective is to either increase throughput by reducing submissions or to comply with maximum job restrictions. Therefore, besides fault tolerance, wrappers do not improve task scheduling and coordination within the allocation.

We have expanded the paragraph that quantifies the queue time issue in climate simulations. Lines 30-34 now read:

However, lately the community has drawn attention to the efficiency of the simulations, taking into account the most demanding part along with the postprocessing, failure, and time spent in the queue. With this in mind, Balaji et al. 2017 proposed a set of performance metrics for Earth system model simulations. Among these metrics, the authors proposed the simulated years per day (SYPD), which is the ratio of the time simulated in years with respect to the runtime of the job in days, as well as the actual simulated years per day (ASYPD), which is the simulated time in years divided by the the time-to-solution of the simulation. Thus, this metric now accounts for time in the queue and also system interruptions.

In Acosta et al. 2024, the authors computed these metrics for 33 CMIP6 simulations executed on 14 machines. Their analysis showed that the difference between ASYPD and SYPD ranged from 0% to 78%. But, they noted that not all institutions reported ASYPD consistently. Some accounted for both interruptions and queue time, while others accounted only for queue time. For those institutions that only accounted for queue time, the spread was between 10% and 20%. The authors therefore concluded ``that queuing time represents an increment of around 10%–20% of the speed of the ESM."

In the Background section, where we introduce the wrappers, we have included their design goal (according to the Autosubmit developers; we do not claim it is our original idea). The wrappers subsection now reads (lines 116–126):

In a shared HPC environment, queuing for resources is ever so frequent (Patel et al., 2020), and users have a limited impact on the priority of their jobs given the importance of fair share.

To reduce the time-to-solution of an Earth System Model (ESM) simulation workflow, the Autosubmit developers came up with a technique called task aggregation or wrapping. Their idea was to increase throughput by avoiding queuing subsequent tasks. For this reason they implemented vertical wrappers, which append workflow tasks into a longer submission.

In addition to vertical wrappers, horizontal wrappers were developed to comply with the platform's policy regarding the maximum number of jobs in the queue.

Finally, there is also the combination of the two types: vertical-horizontal and horizontal-vertical. A vertical-horizontal is made of multiple vertical wrappers running concurrently. Similarly, the horizontal-vertical is a single job made of multiple subsequent horizontal wrappers.

In all wrapper types, the dependencies among the tasks are respected and the underlying application task is not altered by their employment. Tasks are submitted together to the remote platform. Therefore, all steps normally performed, such as saving the restart conditions (or checkpointing), are still executed. Moreover, if a task fails within the aggregated job, Autosubmit will relaunch the failed task without the need of a new job submission.

In this work, we will focus on vertical wrappers, as they are the proposed solution for the long queue times.

In this work, we will focus on vertical wrappers, as they are the proposed solution for long queue times.

Finally, we have further detailed our original hypothesis regarding the vertical wrappers in the introduction (lines 62–64). It now reads:

Although aggregation is utilized in other fields and also by Autosubmit users, with positive impact reported, there is a lack of understanding of the reasons and conditions under which aggregating reduces queue time. Therefore, in this work, we tested whether wrapping subsequent tasks together reduces queue time and if the fair share is the most important factor in reducing queue time.

And we clarified the abstract (lines 2-19). It now reads:

High Performance Computing (HPC) is commonly employed to run high-impact Earth System Model (ESM) simulations, such as those for climate change. However, running workflows of ESM simulations on cutting-edge platforms can take a long time due to the congestion of the system and the lack of coordination between current HPC schedulers and workflow manager systems (WfMS). The Earth Sciences community has estimated the time in queue to be between 10% to 20% of the runtime in climate prediction experiments, the most time-consuming exercise. To address this issue, the developers of Autosubmit, a WfMS tailored for climate and air quality sciences, have developed wrappers to submit multiple subsequent workflow tasks -- the atomic unit of compute in the workflow -- as single submission, without changing them. However, although wrappers are widely used in production for community models such as EC-Earth3, MONARCH, and Destination Earth simulations, to our knowledge, the benefits and potential drawbacks have never been rigorously evaluated. Later, the developers of Autosubmit noticed that the performance of the wrappers was related to the past utilization of the user which reflects on job priority in Slurm via the fair share factor. The objective of this paper is to quantify the impact of wrapping subsequent tasks on queue time and understand its relationship with the fair share and the job's CPU and runtime request. To do this, we used a Slurm simulator to reproduce the behavior of the scheduler and, to recreate a representative usage of an HPC platform, we generated synthetic static workloads from data of the LUMI supercomputer and a dynamic workload from a past flagship HPC platform. As an example, we introduced jobs modeled after the MONARCH air quality application in these workloads, and we tracked their queue time. We found that, by simply joining tasks, the total time-to-solution of the simulation reduces up to 7% with respect to the runtime of the simulations, and we believe that this value is larger the longer the workflow. This saving translates to absolute terms of about eight days less wasted in queue time for half of the simulations from the IS-ENES3 consortium of CMIP6 simulations. We also identified a high inverse correlation of -0.87, between the queue time and the fair share factor in the static experiments.

(1) The manuscript states that the observed 7% reduction in runtime could be "larger in reality". Please expand on the specific real-world factors or complexities (e.g., more dynamic system loads, nuanced fair share policies, or varied backfill algorithm effectiveness) that might contribute to a greater benefit in practice. This would enhance the practical applicability and persuasiveness of the findings.

We thank the reviewer for this constructive comment, and we agree that we should explain precisely why we believe the 7% figure is likely higher in reality.

First, we would like to clarify that the 7% is the maximum difference between the time-to-solution of the unwrapped minus the wrapped workflow divided by the runtime of the workflow. We have observed across all the fair share values, that the wrapped workflow was shorter on average (in terms of time-to-solution) than its unwrapped counterpart, as indicated by the green triangle in Figure 6 that is always positive.

Thus the gains of using wrappers come from 1) the jobs stay about the same or less in queue and 2) there are many times less jobs (if wrappers of 10 tasks are employed in a workflow with 50 sequential jobs, there would be 10 times less jobs submitted to the remote platform).

Therefore, the longer the workflow and the wrappers, the larger should be the gains.

We have rewritten lines 266-273 that introduce the discussion with a clarification of the 7% and also why we believe that it would be beneficial, in general, for longer workflows.

As seen in Figure 6, we achieved a reduction in queue time across all fair share values in the dynamic results. On average, this reduction was 1%, reaching up to a 7% decrease in queue time relative to the total workflow runtime. These results support the hypothesis that the reduction is caused by avoiding multiple submissions.

Since we observed consistent reductions when using aggregation, we anticipate greater gains in longer workflows because longer wrappers can be created, reducing the time spent waiting for resources.

Additionally, the 7% figure could be greater if we consider that the machine had only two days of congestion per week. Current flagship systems are usually congested, and it is not uncommon for jobs to queue for days.

(2) While the Slurm simulator is a strength, a more explicit discussion of its known limitations and how these might influence the generalizability of the results is warranted. For instance, the paper mentions that the simulator "does not have support for dynamic submission times" for constrained jobs as a real Workflow Management System like Autosubmit would. While the authors address this by calculating submission times based on assumed predecessor completion, further detail on the potential implications of this approximation on the reported queue times would be beneficial.

We thank the reviewer for pointing out that the Slurm simulator is a strength of our methodology. We agree that we should be more explicit about its shortcomings.

Therefore, in line 294, we have included a new subsection called "Limitations" in the discussion section that reads:

In this subsection, we discuss the major weaknesses that we identified during our work.

First, the Slurm simulator does not support dynamic submissions, i.e., launching a job the moment its dependency finishes. Therefore, we had to define the submission time in advance by assuming the best-case scenario, in which no task is delayed. Thus, the age factor increases while the job waits in the queue for its dependency to finish, resulting in a higher priority than they would have in reality.

However, with the configuration we tested, we found that the priority added by the age factor was marginal. The maximum time that a job was in the queue in any of our simulations was 1,776 seconds, in the worst fair share case submitted at 15/6/2012 at 20 (submission instant B.3). This adds just 293 (1,176 seconds divided by the total number of seconds in seven days multiplied by 10^5) to the job's priority. This is minimal compared with the priority added by a fair share of just 0.01, which after being multiplied by its corresponding weight of 10^5, adds 1,000 to the job's priority.

Another limitation is that the simulator does not support node sharing among jobs, as is the case in MareNostrum 4. Therefore less than a node requests would be scheduled to whole nodes, wheareas, in MareNostrum 4, they would share resources. But, we have seen systems enforce node exclusivity across the board, as is the case with Lumi.

Finally, the Slurm simulator here employed is not determinist, although the authors of the BSC contribution to it have greatly reduced it (Jokanovic et al., 2018). This is another reason to run multiple experiments and take the average.

(3) The methodology for controlling fair share in static workloads using "dummy" jobs is clear. However, consider adding a brief discussion on whether this method fully captures the complex and dynamic evolution of fair share in a truly live, highly utilized HPC system.

We agree with the reviewer's suggestion to include a discussion of why "dummy" jobs capture the complex dynamics of HPC systems. We expanded the subsection that explains the experimental design of the static workloads (lines 197–201). The paragraph now reads:

In order to control the fair share, given that all the jobs of the workload are submitted at the same time, we preceded the simulation with a batch of ``dummy'' jobs so that all users have usage recorded. Otherwise, all users would have nil usage, therefore maximum fair share, and it would effectively remove the fair share from the scheduling. We set all the users ``dummy'' submission runtime to be proportional to the synthetically generated usage, except for the user employed with launching the workflow, for whom we chose its usage to control its fair share. This was done because we know that users have a recurrent pattern of utilization, as described by Patel et al. 2020. Therefore, the introduction of

dummy jobs made the fair share of the users coherent with respect to the synthetically generated usage.

In the introduction to the static workloads, we also included a reference to a peer-reviewed paper utilizing static traces for performance modeling in HPC. Lines 170–172 now read:

Static workloads are those where all jobs in the description are submitted at the same time. We made this decision to simplify the modeling, since we disregard the complex modeling of the arrival time (Cirne et al., 2000). We chose the number of jobs to be generated so that we stress the system but still within plausible bounds. Moreover, this methodology was also employed by Jeannot et al. 2023 to recreate a HPC environment for performance analysis..

(4) The paper outlines several categories of wrappers (vertical, horizontal, vertical-horizontal, and horizontal-vertical) but focuses solely on vertical wrappers. A brief justification for this specific focus, and perhaps a suggestion for future research avenues exploring the impact of the other wrapper types, would strengthen the introduction or discussion.

We agree that our explanation of why we focused on vertical wrappers was unclear. Therefore, we rewrote the entire subsection about wrappers in the background section (lines 116–126). It now reads:

In a shared HPC environment, queuing for resources is ever so frequent (Patel et al., 2020) and users have a limited impact on the priority of their jobs given the importance of fair share.

To reduce the time-to-solution of an Earth System Model (ESM) simulation workflow, the Autosubmit developers came up with a technique called task aggregation or wrapping. Their idea was to increase throughput by avoiding queuing subsequent tasks. For this reason they implemented vertical wrappers, which append workflow tasks into a longer submission.

In addition to vertical wrappers, horizontal wrappers were developed to comply with the platform's policy regarding the maximum number of jobs in the queue.

Finally, there is also the combination of the two types: vertical-horizontal and horizontal-vertical. A vertical-horizontal is made of multiple vertical wrappers running concurrently. Similarly, the horizontal-vertical is a single job made of multiple subsequent horizontal wrappers.

In all wrapper types, the dependencies among the tasks are respected and the underlying application task is not altered by their employment. Tasks are submitted together to the remote platform. Therefore, all steps normally performed, such as saving the restart conditions (or checkpointing), are still executed. Moreover, if a task fails within the aggregated job, Autosubmit will relaunch the failed task without the need of a new job submission.

In this work, we will focus on vertical wrappers, as they are the proposed solution for the long queue times.