Temporal Classification of Allocations for Reduced Memory Usage

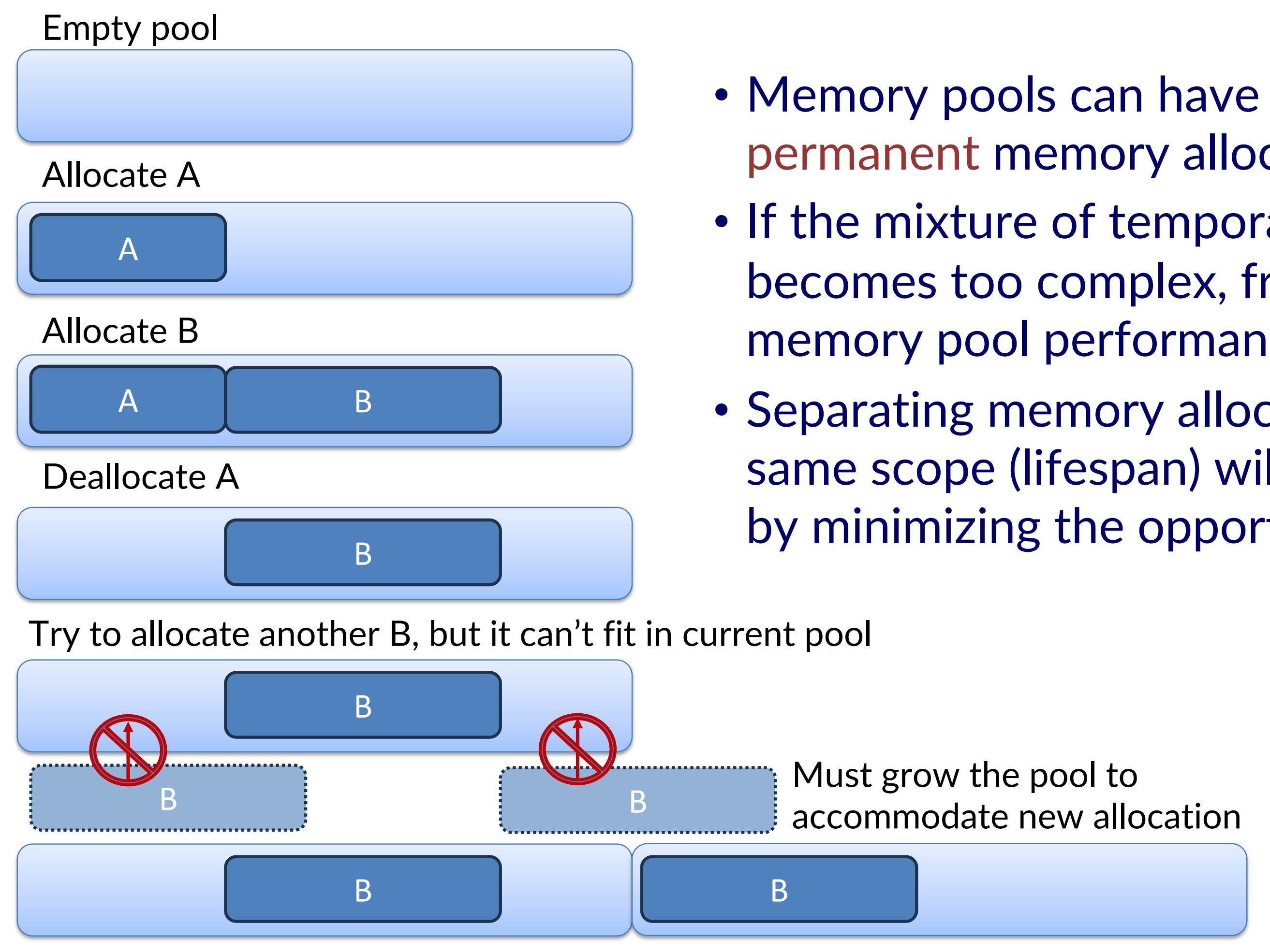
Kristi Belcher[1], David Beckingsale[1], Sam Schwartz[2], Marty McFadden[1]

[1]Lawrence Livermore National Laboratory, [2]University of Oregon

The performance of memory pools in scientific applications varies widely, especially when accommodating for temporary and permanent allocations. The Umpire team from Lawrence Livermore National Laboratory (LLNL) used SAMRAI and BoBa application tests to study how well our machine learning (ML) model could predict these two allocation types. We then use these predictions to create separate memory pools for permanent allocations. We found that separating temporary and permanent allocations into distinct pools reduced peak memory usage significantly (for our tests, by up to 29.5%).

Background

- Umpire (developed at LLNL) provides memory pools which allow a less expensive way to allocate all needed memory for HPC applications, compared to device specific APIs.


Empty pool
Allocate A
A
Allocate B
A
B
Deallocate A
B
Try to allocate another B, but it can't fit in current pool
B
B
B
Must grow the pool to accommodate new allocation
B
B

- Memory pools can have a mixture of temporary and permanent memory allocations.
- If the mixture of temporary and permanent allocations becomes too complex, fragmentation occurs (Fig. 1) and memory pool performance will suffer.
- Separating memory allocations into distinct pools with the same scope (lifespan) will help memory pool performance by minimizing the opportunity for fragmentation to occur.

Figure 1: Depiction of how fragmentation occurs in a memory pool. Because A and B have different lifespans in the pool, when A is deallocated, an additional B allocation will no longer fit in the pool without first growing the pool to accommodate the new allocation, creating fragmentation.

Approach

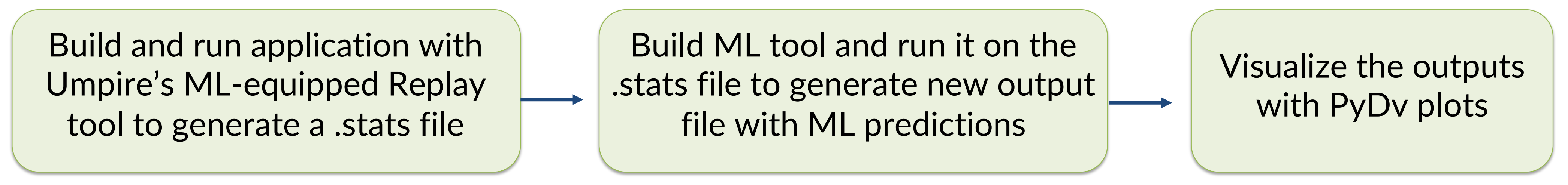
Feature Type	Feature Importance
Encoded Backtraces	73%
Time-Relationship	15%
Allocator Frequency	6%
All Other Features	6%

Figure 2: The types of features we used in our models and their calculated importance.

- We collected different features and used Scikit-Learn to determine their importance (Fig. 2)
- The calculated F1 scores (median value ≥ 0.998) helped us determine that our model was a good fit to the data.

- Our workflow (Fig. 3) consisted of using Umpire's debugging and memory analysis tool, Replay, to trace all memory allocations throughout the SAMRAI and BoBa tests
- The output .stats file from Replay was then used as input to the ML tool to do temporal classifications and categorize the memory allocations
  - These files contain backtrace information, memory addresses, etc. for each allocator throughout the program
- Replay was used again on the output files generated from the ML tool to produce the .ult files for PyDv (an open source, Python based LLNL app) visualization and analysis

Figure 3: Our workflow for generating Replay files that will then be used as input to our ML tool. The resulting file can then be visualized with PyDv for analysis.


Build and run application with Umpire's ML-equipped Replay tool to generate a .stats file

Build ML tool and run it on the .stats file to generate new output file with ML predictions

Visualize the outputs with PyDv plots

Results

- Experiments performed using Replay trace files from BoBa and SAMRAI tests to determine how much memory savings can be had by applying our model.
- SAMRAI test did not use much if any permanent memory, so the ML tool did not categorize allocations to be moved to a separate pool (confirmed by Replay).
- BoBa test used permanent memory which the ML tool correctly classified and put in a separate pool. Fig. 4 shows the resulting memory savings (up to 624 kB saved out of a 2.1 MB total, or about 29.5%).
- Fig. 5 zooms in on the growth of the Permanent Allocations (purple dashed line) as the program starts up.

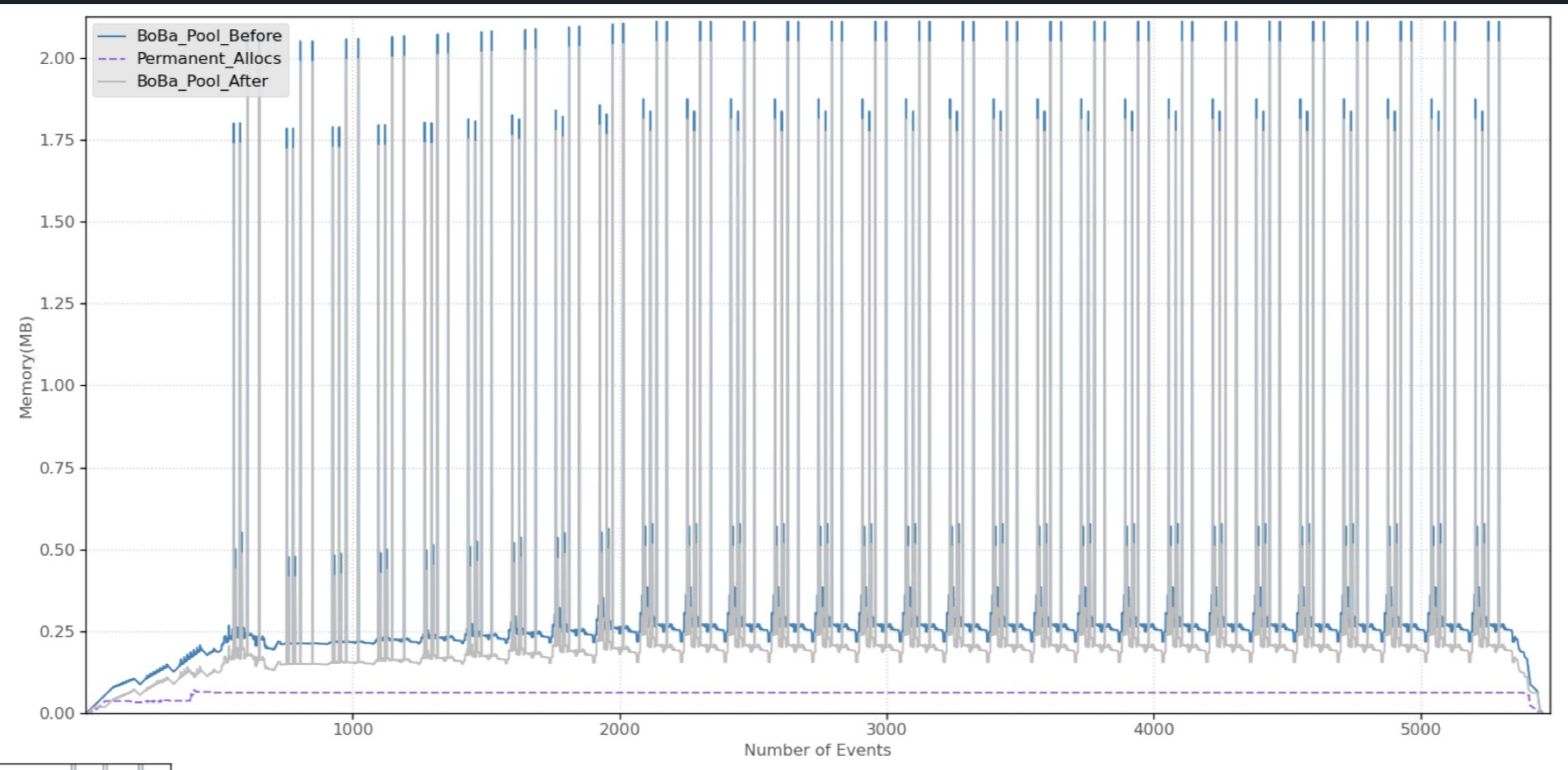


Figure 4: A before and after comparison of the BoBa memory pool with our ML tool. The BoBa memory pool (silver line) is lower after the ML tool predictions compared to before (blue line). The purple dashed line shows how much permanent memory could be separated into a distinct pool.

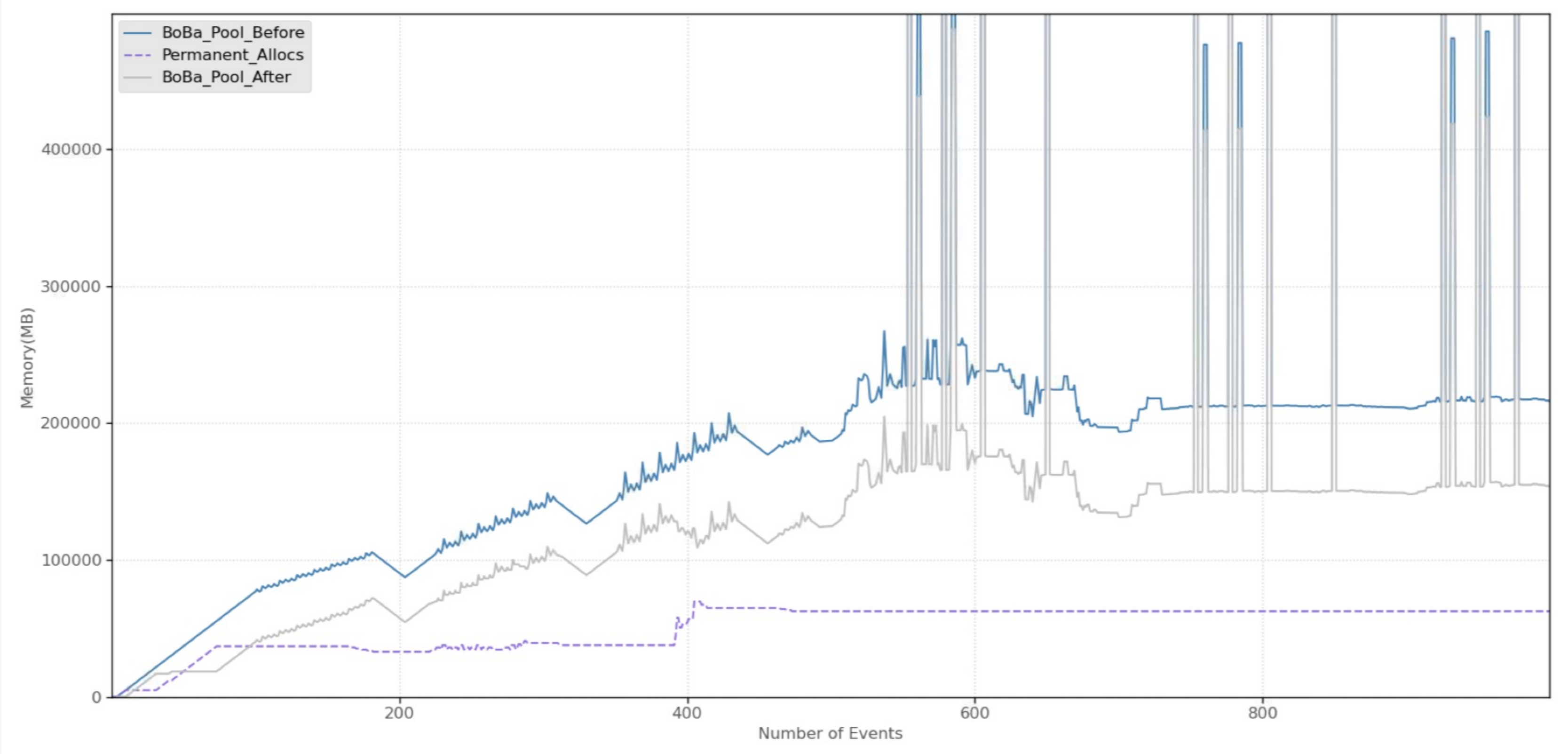


Figure 5: Zooming in on the first 1000 events from Fig. 4, we see the Permanent Allocations (purple dashed line) grow as allocations are made at the beginning of the program.

- In order to see greater memory savings, we will need production level applications which allocate GBs of memory for the duration of the program.
  - These results provide a proof of concept that our ML tool can be used to accurately temporally classify allocations as permanent or not
- Future studies with other, larger applications are expected to be consistent with these results
  - More study needed to determine how this will impact fragmentation within the memory pool and to what degree

Conclusions

Scan the QR code to see our GitHub branch with experiments and reproducibility instructions.

Clone Umpire:
https://github.com/LLNL/Umpire

Contact the Umpire Team:
umpire-dev@llnl.gov

- Using a machine learning model, we can accurately categorize each allocation in a pool as either temporary or permanent.
  - Backtrace information for each allocation is a very important feature
  - Usually, Random Forests were the best prediction model
- Using this strategy led to a reduction of peak memory usage by up to 624kB out of the 2.11MB total (29.5%) for the BoBa application test.
  - This leads us to believe fragmentation was reduced, but future study will be needed to determine by how much and if that effects overall performance
- Regarding future work, to see bigger memory savings, we need to use applications with larger, more complex memory allocations.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory.