USENIX ATC '21

Ayudante: A Deep Reinforcement Learning Approach to Assist Persistent Memory Programming

Hanxian Huang, Zixuan Wang, Juno Kim, Steven Swanson, Jishen Zhao

University of California, San Diego

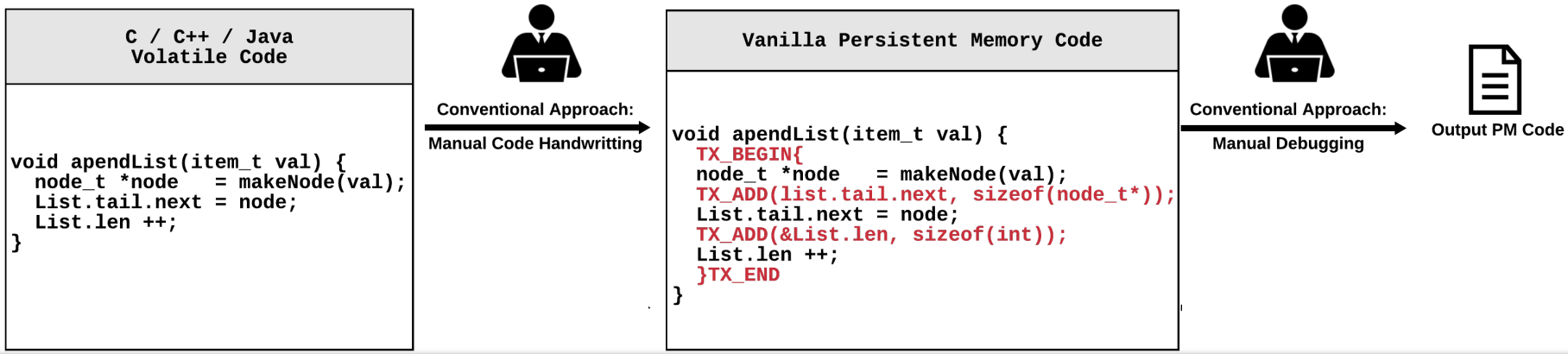
hah008@ucsd.edu

Outline

- Background and Motivation
- Ayudante Framework
- Evaluation Results
- Conclusion

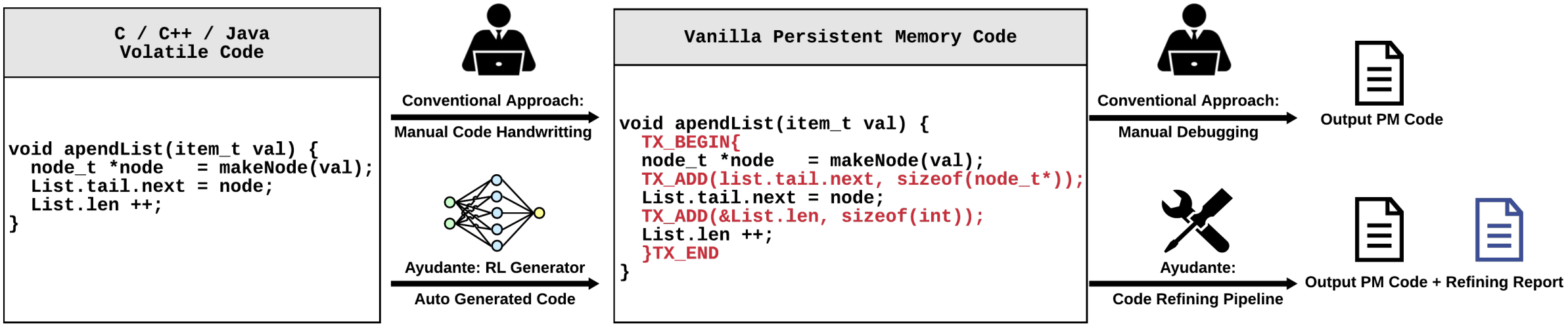
Background and Motivation

- Challenges in persistent memory (PM) programming
  - Non-trivial labor effort
  - Error-prone
  - Require a good knowledge of PM
- Our goal
  - Assist PM programming by transforming volatile memory-based code into corresponding PM code with minimal programmer interference



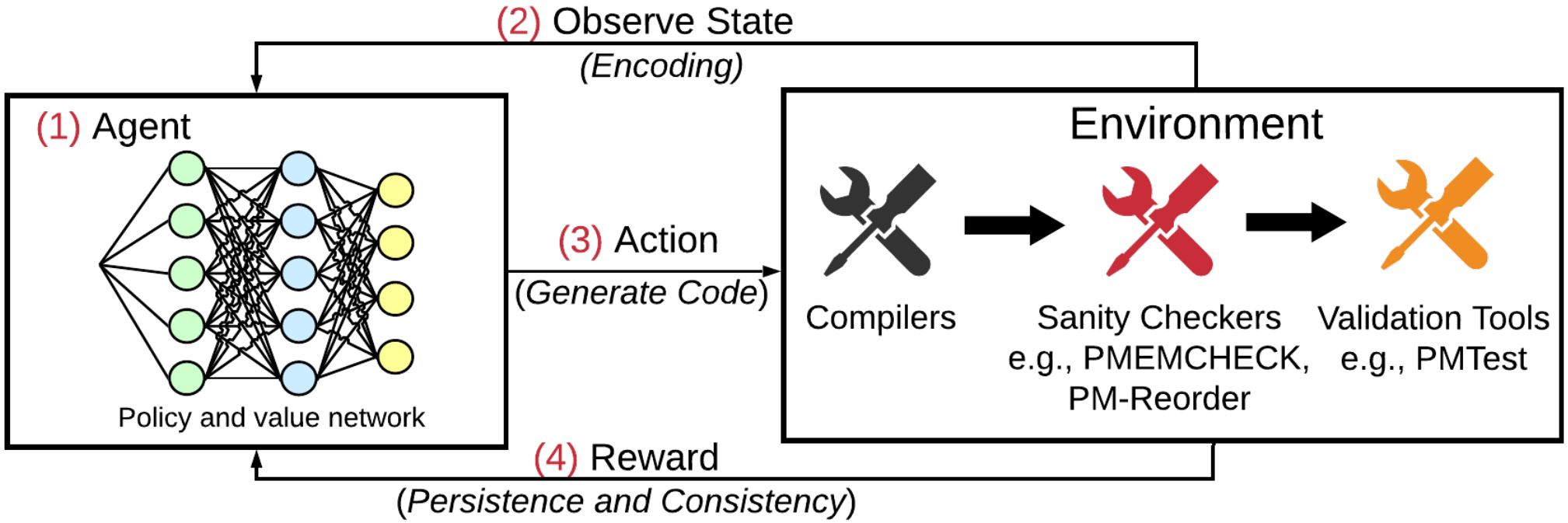
Ayudante Framework

- Two key components
  - 1. Deep reinforcement learning (RL)-based code generator
  - 2. Code refining pipeline



Deep RL-based Code Generator

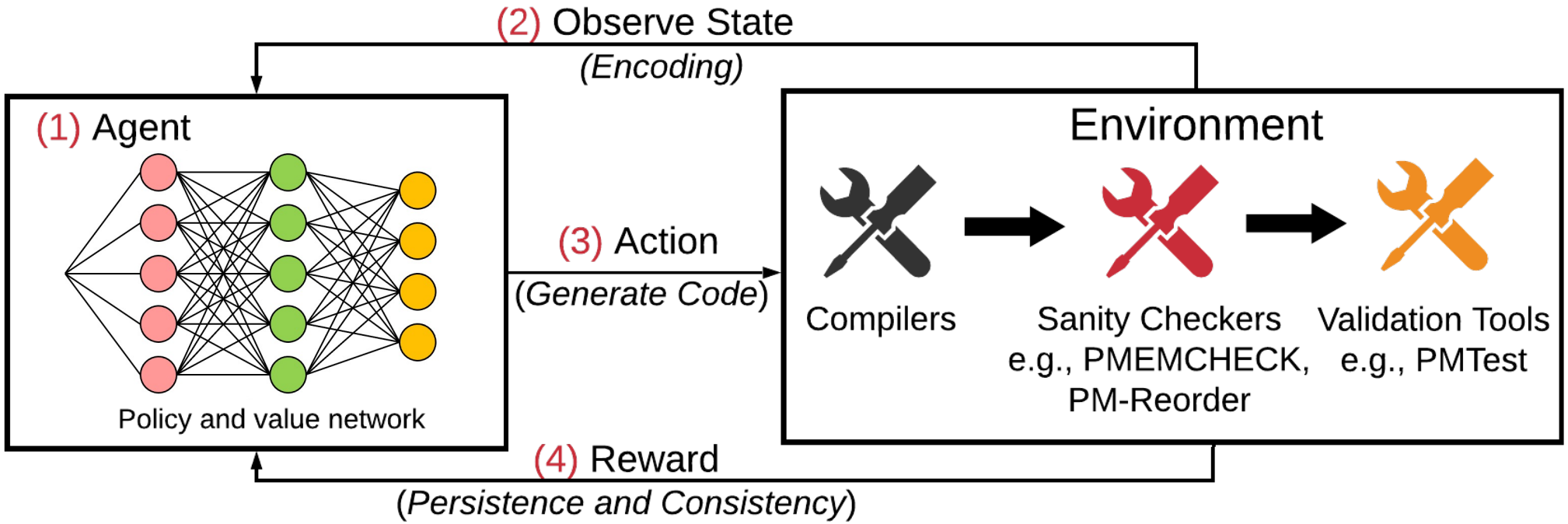
- Agent
- State
- Action
- Reward



Deep RL-based Code Generator

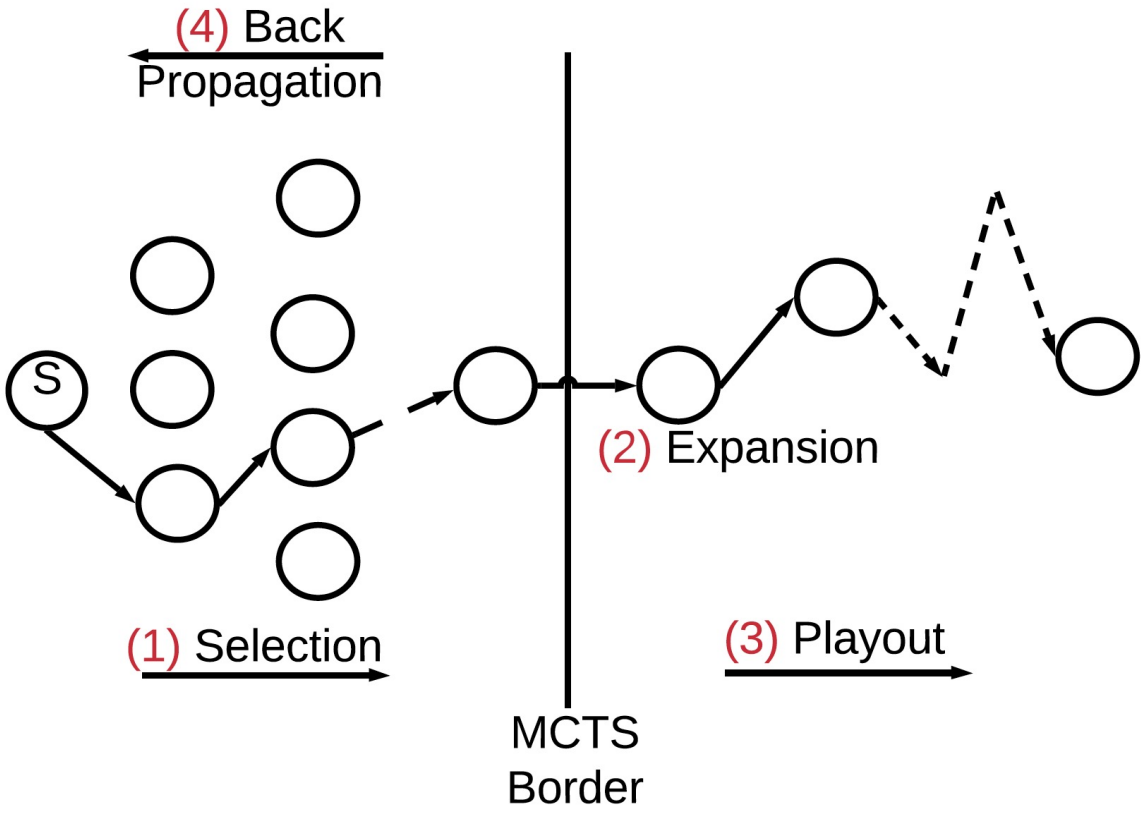
- Agent: the executor of actions on a state
- State: <string, position>
- Action: navigation or edit
- Reward: $\phi_1 \cdot lnS + \phi_2 \cdot lnM + \sum_{\{i=1\}}^{n} \rho_i E_i$
  - ϕ_1, ϕ_2, ρ_i are penalty factors
  - S, M, E_i #steps, #modifications and #errors reported by PM checker i

Deep RL-based Code Generator – the Model



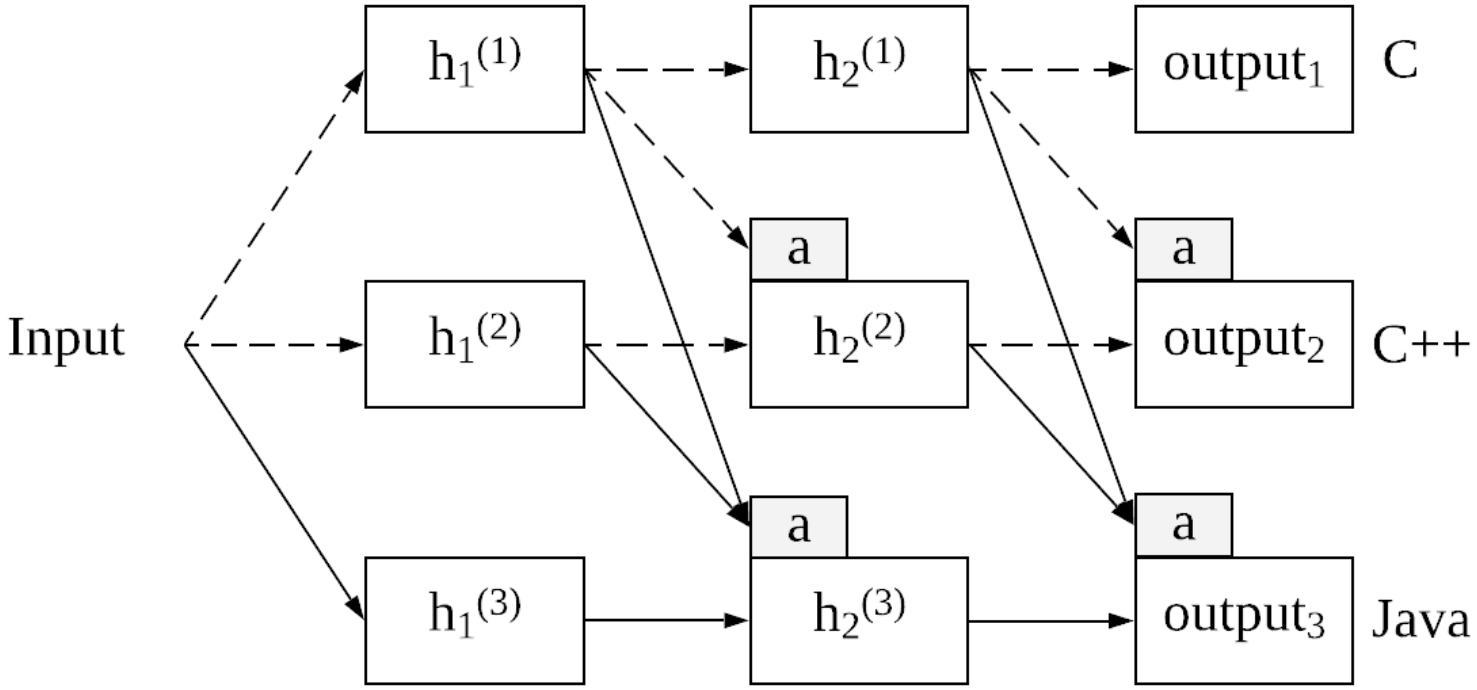
Deep RL-based Code Generator – Search Strategy

- Monte-Carlo Tree Search



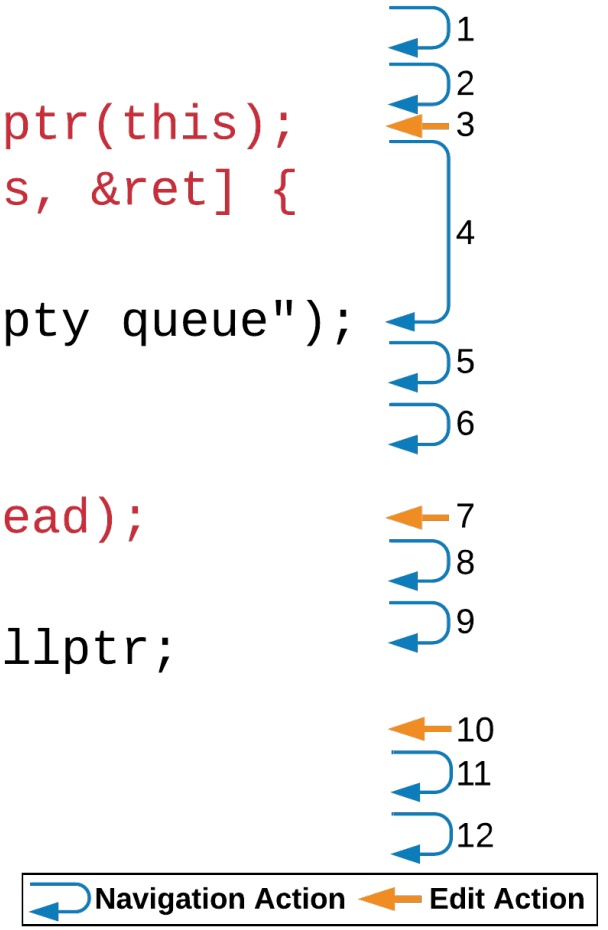
Deep RL-based Code Generator – Transfer Learning

- Transfer Learning: From C/C++ to Java



An Example of RL-based Code Generation

```
1  int64_t Queue::pop(){
2    int64_t ret = 0;
3    auto pool = pmem::obj::pool_by_vptr(this);
4    obj::transaction::run(pool, [this, &ret] {
5      if (head == nullptr)
6        throw std::runtime_error(”Empty queue");
7      ret = head->value;
8      auto n = head->next;
9      obj::delete_persistent<Node>(head);
10     head = n;
11     if (head == nullptr) tail = nullptr;
12   });
13   return ret;
14 }
```



Ayudante Framework

- Two key components
  - 1. Deep reinforcement learning (RL)-based code generator
  - 2. Code refining pipeline
    - (1) Compilers
    - (2) Sanity checkers: PMEMCHECK and PM-Reorder
    - (3) Validation tools: PMTest, XFDetector, and AGAMOTTO

A refining suggestion report example

Vanilla generated code

```
int node_construct() {
// set up btree node
...
pmemobj_persist(pop, node,
                a->size);
// persist data
...
}
```

Call

```
void btree_insert() {
// set up btree stucture
...
POBJ_ALLOC(..., node_construct);
pmemobj_persist(pop, dst, args.size);
// duplicated persistence
...
}
```

Improved code

```
int node_construct() {
// set up btree node
...
pmemobj_persist(pop, node,
                a->size);
// persist data
...
}
```

```
void btree_insert() {
// set up btree stucture
...
POBJ_ALLOC(..., node_construct);
...
}
```

Implementation

- Training dataset
  - Volatile version PMDK example codes

- Testing dataset
  - Microbenchmarks: array, string, list, queue, btree, rbtree, hashmap
  - KV store application
  - Open source leetcode solution

Evaluation

- PM Checker Passing Rate (CPR) in Inference

Testing Set	Checkers in Environment	CPR	LOC
Microbenchmarks and KV store application	PMEMCHECK	87.5%	12.3%
	PMEMCHECK & PM-Reorder	**100%**	13.4%
	PMEMCHECK & PM-Reorder & PMTest	**100%**	13.8%
Leetcode solution	PMEMCHECK	60.2%	12.5%
	PMEMCHECK & PM-Reorder	62.1%	13.1%
	PMEMCHECK & PM-Reorder& PMTest	78.7%	13.4%

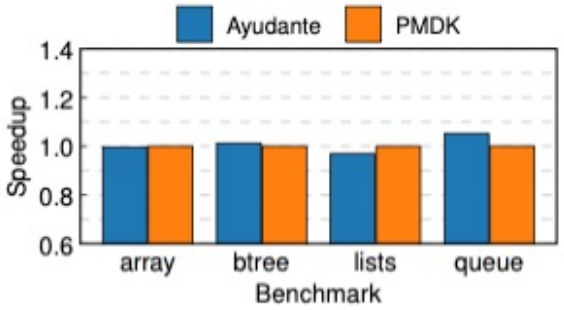
Evaluation

- Labor Effort Reduction
  - Lines of code (LOC) changed

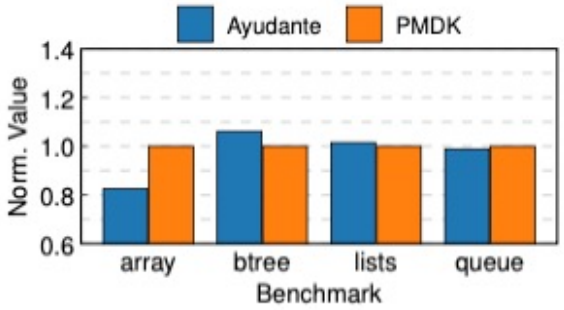
Testing Set	Checkers in Environment	CPR	LOC
Microbenchmarks and KV store application	PMEMCHECK	87.5%	12.3%
	PMEMCHECK & PM-Reorder	**100%**	13.4%
	PMEMCHECK & PM-Reorder & PMTest	**100%**	13.8%
Leetcode solution	PMEMCHECK	60.2%	12.5%
	PMEMCHECK & PM-Reorder	62.1%	13.1%
	PMEMCHECK & PM-Reorder& PMTest	78.7%	13.4%

Evaluation

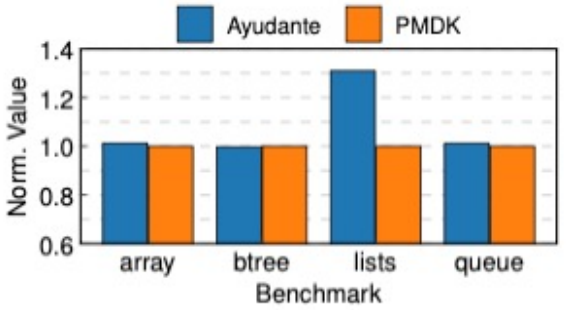
- Execution Performance on an Intel Optane DC PM server



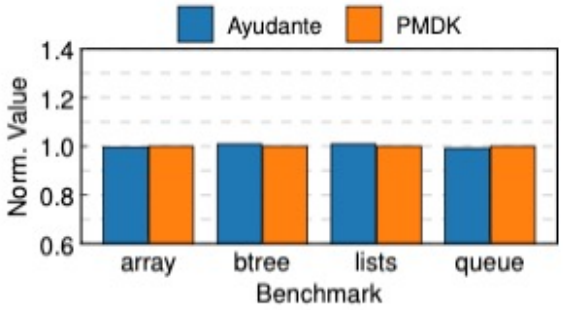
(a) Bandwidth



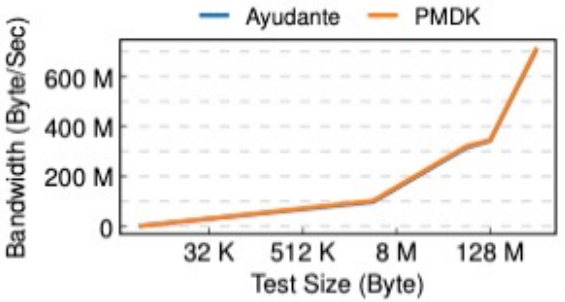
(b) LLC load



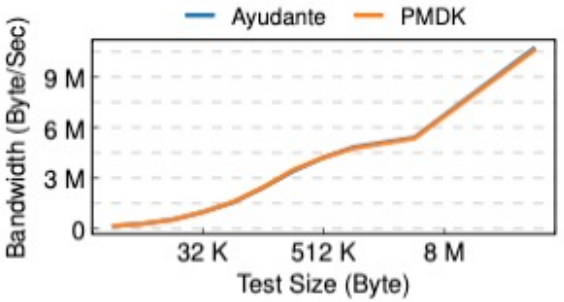
(c) LLC store



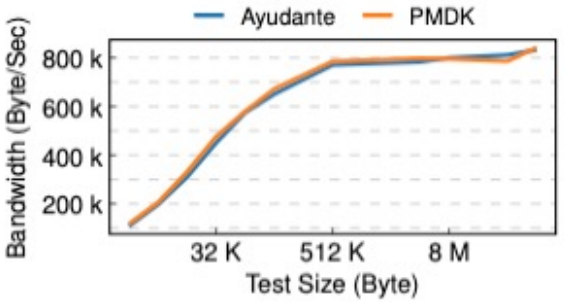
(d) dTLB load



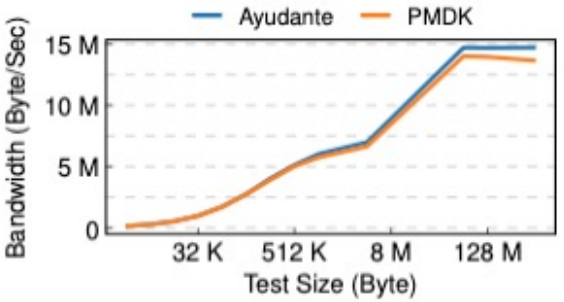
(a) Array



(b) B-Tree



(c) Lists



(d) Queue

Conclusion

- Ayudante offers
  - **Efficient PM code generation** through a deep RL model augmented with Monte-Carlo tree search
  - **Reduced bugs** through a deep RL model pre-trained to avoid bugs detectable by checkers in the training environment
  - **Code refining reports and improved performance** through a code refining pipeline

Reduce programmer's burden on PM programming

USENIX ATC '21

Ayudante: A Deep Reinforcement Learning Approach to Assist Persistent Memory Programming

Hanxian Huang, Zixuan Wang, Juno Kim, Steven Swanson, Jishen Zhao

University of California, San Diego

hah008@ucsd.edu