YODA: A Pedagogical Tool for Teaching Systems Concepts

Apan Qasem
Texas State University
San Marcos, TX, USA

ABSTRACT

Computer science undergraduates often struggle in hardware-oriented courses like Computer Organization and Computer Architecture. Active learning instruments can improve student performance in these classes. Regrettably, few tools exist today to support the creation of active learning teaching material for such courses.

This paper describes YODA, a pedagogical tool for creating active learning content to help teach systems concepts. At the core, YODA is a collection of functional simulators embedded into a custom Jupyter kernel. YODA produces notebooks that allow students to learn about a system through guided interaction and observation. We have been using YODA at our home institution for two years and have seen significant improvement in student learning outcomes.

CCS CONCEPTS

• **Social and professional topics → Student assessment**;

KEYWORDS

active learning, computer architecture, pedagogical tools

ACM Reference Format:
Apan Qasem. 2022. YODA: A Pedagogical Tool for Teaching Systems Concepts. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022), March 3–5, 2022, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3478431.3499322

1 INTRODUCTION

Computer science students often struggle in hardware-oriented courses like Computer Organization and Computer Architecture [1, 14, 17]. Students are either not learning as much as the instructors expected or are generally earning poor grades. The situation is aggravated when students take these courses in the first two years of study. Consider for example, the student grades in introductory hardware and programming courses at our home institution [1] On average students earned a better grade in the programming courses (+ 0.15). What is more concerning however, is that this disparity is more prominent by gender and race. The performance gap between hardware and programming courses is greater for female students than their male counterparts and for non-white students than white.

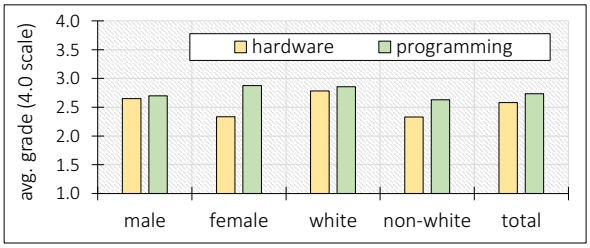

Figure 1: Average student grades in hardware and programming courses during 2019-20 (521 students)

Many factors contribute to this unsatisfactory outcome. CS students are often not invested enough in the hardware courses because they do not perceive them as contributing to their career as a software engineer or programmer. The proliferation of high-level programming frameworks and sophisticated IDEs have reduced the need for a deep understanding of the underlying systems and as such many recruiters downplay the importance of exposure to hardware concepts. Another possible reason is that these courses entail a lot of *reading* and students learn about systems at an abstract level. Some departments, will offer a course in Digital Logic in which students get some hands-on experience working with a micro-controller board. But in general it is infeasible to get students to work on real hardware to illustrate concepts in processor and memory design.

Active learning methods have been shown to be effective in improving student learning outcomes in a variety of CS courses [8]. But its application in hardware and systems courses is somewhat limited.

Learning by doing: The best way to learn a system is to build your own. This approach works well for capstone courses like Compilers and Operating Systems. But introducing such a project in lower-division courses is problematic. First, because the curriculum is already *saturated*, there is often not enough time to introduce such a project without an overhaul of the existing syllabus. Second, in many programs students take the Organization course in their second or third semester when they do not have the requisite programming background to take on a project of this scale.

Simulators: Active learning content can be created around simulators. But this approach also has limitations. Simulators are typically designed for research and thus try to be as faithful as possible to the underlying systems they represent, This creates a usability issue for lower-division courses. For example, GEMS [9] is the most efficient and accurate processor simulator out there but is difficult to use in a classroom because of a steep learning curve. MARS, which is designed for teaching [19], is used in architecture courses that follow the MIPS edition of the P&H text [12]. But its focus is solely

[1] doctoral granting R2 institution

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA

ACM ISBN 978-1-4503-9070-5/22/03…$15.00
https://doi.org/10.1145/3478431.3499322

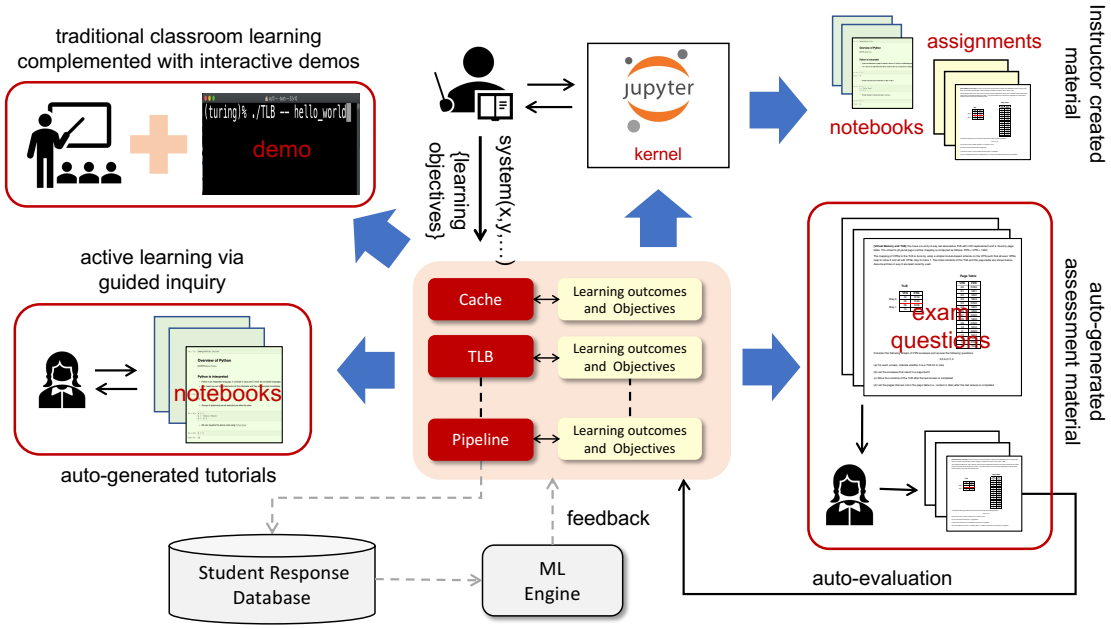

Figure 2: YODA pedagogical workflow. The machine learning component, outlined in dashed gray lines, has not been tested in a classroom setting and is not the subject of this paper. We include it here to show future direction.

on the processor (i.e., Chapter 4 of [12]) and thus is not suitable for the other systems covered in that class.

This paper presents YODA, a pedagogical tool for integrating active learning content in lower-division Organization and Systems courses. YODA is inspired by Jove, a tool for teaching theory concepts to CS undergraduates [5]. The key idea is to give students the opportunity to interact with a system via guided inquiry. The interactions happen in a notebook environment and is structured as a series of query-response events. The queries and responses are bi-directional. The student can query the system and get a response, and *vice versa*. YODA follows the *many small programs* principle [2]. Activities are short, focused and repeatable (via variants).

At its core, YODA is a collection of functional simulators with the following distinguishing features.

- The simulators are designed with teaching in mind. As such, they distill the core functionality of a system and abstract out many of the details that would need to be addressed in a full-blown simulator.
- YODA is integrated with Jupyter and provides a simple, and familiar, interface for guided inquiry.
- YODA comes with a set of active learning content for each system. Instructors can select activities from the database, extend and modify them or create new ones
- A set of learning outcomes is pre-programmed into YODA. The built-in algorithms in YODA can *automatically* generate various types of assessment material corresponding to a specific set of learning outcomes. YODA also allows instructors to add new learning outcomes and create their own assessment material

The rest of the paper is organized as follows. Section 2 describes the YODA architecture and workflow. In Section 3 we present an example teaching module designed around YODA. We present evaluation results in Section 4. Related work is discussed in Section 5. Finally, in Section 6, we reflect on our experiences using YODA and discuss futures plans.

2 YODA WORKFLOW

Fig. 2 gives an overview of the YODA architecture and shows how it supports the pedagogical workflow. The system simulators are implemented in a Jupyter *kernel*. Instructors can use YODA in two modes. They can simply instantiate a YODA kernel in the Jupyter environment and create their own teaching and learning materials in the form of Jupyter notebooks. Alternatively, they can directly connect to the simulators via a web interface and ask YODA to generate custom learning materials from its internal database.The generated material come in the form of editable notebooks and thus the instructor can further customize them if needed. When creating learning material via the web interface, the instructor is presented with a set of options including types of learning material to be created and the topics to be covered. The YODA source is available for free to educators[2]. Users can add new learning outcomes, teaching materials and even new simulators using the YODA C++ API.

2.1 Simulators in Jupyter Kernels

To date, we have implemented three simulators in YODA: (i) cache (ii) VM and TLB and (iii) 5-stage pipelined CPU (*a la* MIPS). Each kernel mimics the behavior of the corresponding system in the

[2]not open-source because of the embedded assessment material

Jupyter notebook environment. Each system supports the following core commands.

- `instantiate(configuration,...)`: create the system with the specified configuration
- `populate(data...)`: populate system with data and/or instructions
- `input(x,...)`: send `x` into the system; for cache and TLB input represents a CPU request for data while for the processor it represents a MIPS instruction.
- `output(options...)`: display system state

The above commands are the minimum required to support the query-response tutorials. Additional commands are available for each simulator to simulate more complex behavior.

2.2 Learning Outcomes

YODA includes a set of pre-defined learning outcomes and objectives for each supported system. The learning outcomes are drawn from ACM2013 [16]. Each learning outcome is broken down into a set of objectives. The objectives are then mapped to specific system behavior. The learning material database in YODA is indexed by a key comprising of an objective-behavior pair. When YODA receives a request to generate learning material, an algorithm first maps the objective to the specific system behavior and then retrieves the material using the objective-behavior key.

2.3 Learning Material

2.3.1 Interactive demo. These notebooks are designed to illustrate system behavior. They are intended to be used by the instructor as an in-class demo in which the instructor steps through a sequence of commands and discusses various aspects of the system. Additional instructional material (e.g., slides) can be seamlessly interleaved in the demo.

2.3.2 Self-paced tutorials with guided inquiry. This constitutes the bulk of the teaching material. Each activity maps to an objective in the database and has the following structure

- *Introduction:* text and figures explaining some aspect of system behavior; hard-coded in YODA.
- *Illustration:* explanation of concept with examples. Students are asked to input a value into the system and observe its behavior. YODA can create many examples for the same behavior. Students can also create new examples as they are working through the tutorial.
- *Challenges:* a series of query-response exercises in which students are asked to determine system output for a given input. YODA provides feedback when student enters an incorrect response. The questions are randomly selected at the time the tutorials are generated. This ensures that two students working on the same activity, or a student repeating a tutorial will likely see different variants of the same query.

2.3.3 Assessment. The summative assessment material are extended versions of the challenges that a student encounters in the tutorials. They are designed to evaluate one specific learning outcome. In most instances, students are asked to present the final state of a system after it has processed a sequence of inputs. The questions are designed in a way such that the students can arrive at the correct solution only if they understand all aspects of the system behavior.

YODA allows instructors to alter the input fields and system parameters to generate many variants of the an assessment question with the same degree of difficulty. We found this to be a useful feature to curb academic dishonesty. The default format of the assessment material are Jupyter notebooks that students can submit via Google Collab [4]. The instructor can also generate PDF or other formats for a paper-based exam, if desired.

Table 1: TLB topics covered in YODA module

	Topics / Objectives
1.	memory address space and the need for virtual memory
2.	virtual to physical address translation, page numbers and offset
3.	modulo-based mapping
4.	the Page Table and the TLB
5.	TLB hit, TLB miss, Page Table Hit, PageFault
6.	data locality and types of TLB misses
7.	Set associativity
8.	LRU replacement policy

3 YODA MODULE: THE TLB

To date, we have developed and implemented four teaching modules around YODA.

- Cache organization
- Virtual memory and TLB organization
- Processor datapath
- Pipelining and hazard detection

In addition to the instructional material generated by YODA, each module includes a slide deck, reference material and pedagogical notes. In the interest of space, we only discuss the TLB module in this section. The TLB is a crucial component of modern computer architecture. It is included as a core topic in Computer Architecture courses. Understanding how the TLB works is challenging for students as it requires them to think in multiple layers of abstraction.

3.1 Learning outcomes

ACM2013 lists TLB in the Architecture Knowledge Area in the core Knowledge Unit AR/Memory System Organization and Architecture. However, it is only listed as a topic with no associated learning outcome. Thus, we use the following: *Understand how the TLB works and the role it plays in paging and memory management.* We break down the above into the set of objectives shown in Table 1.

3.2 Simulation Engine

As discussed previously, in designing the simulators, we attempt to simplify the behavior as much as possible while retaining the core functionality. For the TLB, we exclude features like multiple levels and separation of instruction and data entries. Since the TLB cannot be explained without the Page Table, the simulation engine includes a Page Table module. Table 2 shows the control parameters of the TLB simulator. The instructor can adjust the listed parameters to

Table 2: Parameters to control TLB simulation

System Attribue	Control Parameter
TLB capacity	number of sets
TLB associativity	direct-mapped, 2, 4, 8-way, full
TLB replacement policy	LRU, MRU, random
Page Table capacity	number of entries
Pages on disk	number of PT entries on disk
Address length	4, 8 or 16 bits
VPN to PPN mapping function	VPN + N, N supplied by user

instantiate a TLBs different configurations which can be integrated into the learning and assessment materials.

3.3 In-class Demo

The in-class demo is a step-through of the TLB simulator which can be done either in a Jupyter notebook or inside a terminal window. The demo shows step-by-step how the contents of the TLB and the Page Table change when processing a sequence CPU requests. The default configuration for the demo uses a 2-way TLB with two sets per way and a page table with 16 entries. we found this configuration to be suitable for an in-class demo that lasts about 30 minutes.

The TLB and the Page Table are populated prior to the start of the demo. The LRU entries in the TLB are marked and four entries in the Page Table are left on the disk. YODA selects the memory references such that the simulation contains at least one TLB hit, two TLB misses and one page fault. Further, it ensures that one of the TLB misses is a cold miss while the other is due to the replacement policy.

During the demo, the instructor inputs the generated CPU requests one-by-one, displays the contents of the TLB and Page Table after each entry and explains the behavior. The demo can be made interactive by asking students to *guess* the behavior. Time permitting, a more complex configuration with an extended sequence of memory references can be used in which students participate in the activity in teams.

3.4 Self-paced Tutorials

For this module YODA creates eight *baseline* interactive tutorials covering the set of topics listed in Table 1. In addition, we have a *capstone* activity that ties in all concepts and does an end-to-end walk-through of the simulator.

In general, it is best if students work through activities in the prescribed sequence. However, students can also work on certain activities independently and out of order. For example, the tutorial associated with modulo-based mapping can be done at the very beginning. The instructor may choose to use a subset of activities or splice together tutorials from other modules. For instance, if students already have a good understanding of caches then set associativity and LRU can be skipped and students can directly move to the *capstone* activity. On the other hand, the set associativity and LRU activities are written in a way such that they can be readily embedded in a module on caches.

3.5 Assessment

In addition to the questions embedded in the tutorials, YODA generates several forms of assessment for this module. Short answers / multiple choice questions can be extracted from each tutorial. The main learning outcome is assessed using a long-form question. In this question, the students are given the configuration and the contents of a TLB and a Page Table, along with a string of CPU requests. They are then asked to label each CPU reference as TLB hit, TLB miss or a Page Fault. In addition then need to fill out two tables showing the final contents of the TLB.

By default, YODA will generate questions that evaluates students understanding of each objective. However, the instructor can configure the question to test a subset of outcomes. For example, the TLB can be made direct-mapped such that students are not tested on their understanding of set associativity and LRU policy.

4 EVALUATION

We have been using YODA at our home institution for four semesters, starting Fall 2019. Each semester we have attempted to revise and improve the tool based on student feedback and our own experience in using it. To get a consistent picture, we have introduced the YODA teaching materials in only the undergraduate Computer Architecture course. This is a sophomore level course that students typically take in their fourth semester. It is a required course for both the BA and BS degrees in CS and has CS2 as a pre-requisite. The course uses the MIPS edition of the Patterson and Hennessy textbook [12]. Multiple sections of the course is offered every semester. In each semester, YODA was used in only one section, which served as the *intervention* group. We selected another section, taught by the same instructor in the same semester, as the *control* group.

4.1 Learning Outcome

For each learning outcome associated with each YODA module, we included one assessment question in the final examEach question was mandatory and constituted 10% of the final exam grade. The Cache and TLB modules have been used in all four semesters between FA19 and SP21, the Processor Datapath in three semesters between SP20-SP21 and the Pipelining module only in SP21. Figs. 3-5 show assessment data for the four modules. The figures show the letter grade the students earned on the assessment question and the *passing rate*, which is calculated as the percentage of students earning C or better. We only count grades for students who actually show up for the final exam. Therefore, the passing rate for the assessment question is higher than the overall class average. Data is shown for both the intervention (YODA) and control sections. Overall, the higher passing rates in the intervention sections suggest that the guided inquiry materials were effective in improving student learning outcomes. Next, we discuss our experience and try to explain some of the observed patterns.

Setbacks in the initial release. The first introduction of the YODA modules in FA19 did not go smoothly. There were bugs in the system that caused occasional crashes and prevented the students from finishing some tutorials in time. We learned from our initial mistakes. We fixed the bugs in YODA and improved several aspects including simplifying the language in the guided inquiry activities and setting up JupyerHub locally on our institution's server.

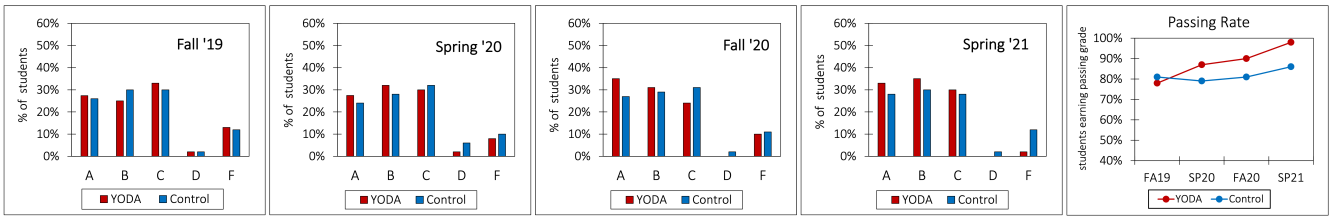

Figure 3: Student grade distribution and passing rates. Cache module. FA19-SP21

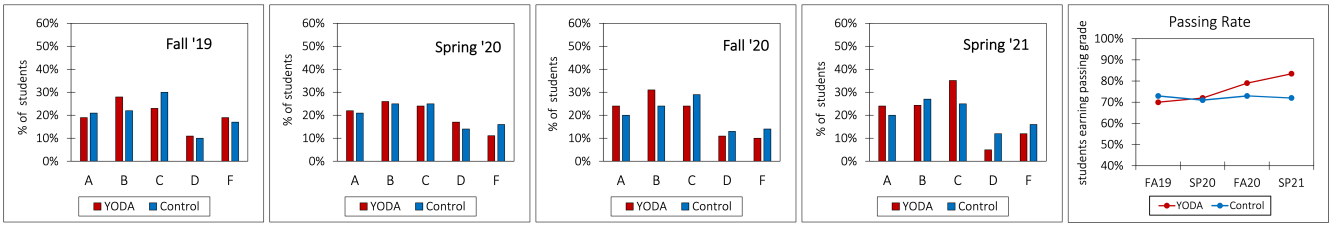

Figure 4: Student grade distribution and passing rates. TLB module. FA19-SP21

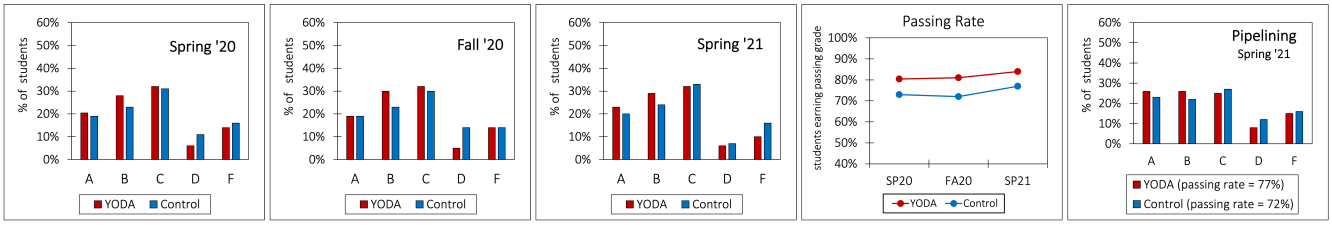

Figure 5: Student grade distribution and passing rates. Processor Datapath and Pipelining modules. SP20-SP21

In FA19, the control group actually did better, although marginally so, than the YODA group. We see the picture change in subsequent semesters in which a relatively higher percentage of intervention students earned A's and B's and also had an overall higher passing rate. Furthermore, the first implementation of the Processor Datapath also went much better as indicated by the outcome results from SP20.

Mandatory assignment Another factor that contributed to the subpar performance in FA19 was that we made the tutorials optional. Only about half the students in the class completed them fully. We made the activities quasi-mandatory starting from SP20. Students who completed all activities, were allowed to drop their lowest quiz grade which constituted 0.5% of the cumulative grade. Although the reward was minimal, we saw a sharp increase in participation rate, reaching close to 90%.

The COVID effect The improved grades in the assessment questions cannot be attributed to YODA modules entirely. FA20 and SP21 classes were conducted online due to COVID. Since the face-to-face contact hours were reduced, we felt the self-paced tutorials had higher utility for students in those semesters. The learning outcome grades may decrease, perhaps to SP20 levels, when instruction modality rolls back to face-to-face in FA21.

Individual and qualitative analysis In analyzing the scores at an individual level, we found that YODA can be a boost for the weaker students. On average over 70% of students that had a C or D average in all *other* graded material, received an improved *score* on the YODA module than their class average. By contrast only about 7% of B students earned a superior score. Our own interactions with the students corroborate these numbers. In most instances, the students who told us they found the tutorials helpful were the ones who were *lost* at the start of the semester.

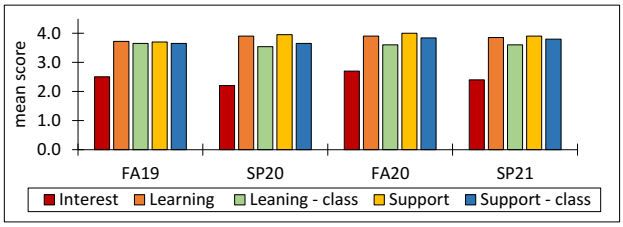

Figure 6: Student perception of YODA modules

4.2 Student Perception

In each class where a YODA module was introduced, we conducted an end-of-the semester survey to gauge student interest in the module topic and assess student perception of the learning experience.

Questions were selected from the Student Assessment of Learning Gains (SALG) survey. Students were asked to rate the module in three categories (i) learning support (ii) learning experience and (iii) confidence and interest gains. Students answered each question on a scale of 0-4 (e.g., strongly disagree, disagree, neutral, agree and strongly agree).

Fig. 6 shows the results of the survey for the Cache and TLB modules from FA19 to SP21. The participation rate on the survey was 82%. In the regular student evaluation there are two questions that relate to student learning experience and learning support. For comparison, we include the scores on these question as well. The students rated their learning experience and support for learning quite highly. The scores in both categories improved slightly from FA19 to SP20 but we do not observe any improvement beyond that. Comparing the scores in these two categories with the scores in the overall class evaluation, we see that students rated their learning experience and support in the modules slightly better than their experience in the overall class. Although the increase is small, we consider the numbers to be meaningful. The student confidence gains numbers were consistently lower than the other scores, which was disappointing. Although the grades of the student learning outcomes suggest that most students did not particularly struggle with the material, this was not reflected in student perceptions.

5 RELATED WORK

Active Learning. There have many efforts at integrating active learning activities in undergraduate computer science courses. These include role playing exercises [10], peer instruction [3, 13], pair programming [11], co-operative learning [3, 7], use of classroom technology such as tablets [15]. Understandably, a vast of majority of these efforts focus on introductory programming courses. Notable efforts in non-programming courses include work by Porter *et al.* who implement peer instruction in Computer Architecture and Theory of Computation courses [13]. Ham and Myers introduce cooperative guided learning in a Computer Organization course [7]. We believe the YODA is complementary to these prior efforts.

Simulators Simulators have been used in many undergraduate computer architecture and organization courses [6]. Many of them use the SPIM or the MARS simulator associated with MIPS architecture, which is widely used. There have also been attempts to design simulators specifically for the classroom [18].

6 CONCLUSIONS AND FUTURE WORK

In spite of the disruptions caused by COVID, our experience in using YODA over the last two years has been quite positive. The data collected so far shows that incorporating YODA-created active learning material can improve student learning outcomes in lower-division hardware-oriented courses.

Thus far, we have only evaluated YODA in the Computer Architecture course. Our future plans include creating more teaching modules around YODA and integrating the materials in a Computer Organization course at our home institution. We also hope to encourage faculty at other institutions to adopt the active learning materials for their classes.

ACKNOWLEDGEMENT

This work was supported by the National Science Foundation through award OAC-1829644, and equipment grants by IBM and NVIDIA.

REFERENCES

[1] Nevena Ackovska and Sasko Ristov. 2013. Hands-on improvements for efficient teaching computer science students about hardware. In *2013 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 295–302.

[2] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An analysis of using many small programs in cs1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 585–591.

[3] Joe D Chase and Edward G Okie. 2000. Combining cooperative learning and peer instruction in introductory computer science. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*. 372–376.

[4] Google. [n.d.]. Google Colab. https://colab.research.google.com.

[5] Ganesh Gopalakrishnan and Rick Neff. 2021. Automata and Computability Education via Jove. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 1374–1374.

[6] Herbert Grunbacher. 1998. Teaching computer architecture/organisation using simulators. In *FIE'98. 28th Annual Frontiers in Education Conference. Moving from'Teacher-Centered'to'Learner-Centered'Education. Conference Proceedings (Cat. No. 98CH36214)*, Vol. 3. IEEE, 1107–1112.

[7] Yeajin Ham and Brandon Myers. 2019. Supporting Guided Inquiry with Cooperative Learning in Computer Organization. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 273âĂŞ279. https://doi.org/10.1145/3287324.3287355

[8] Qiang Hao, Bradley Barnes, Ewan Wright, and Eunjung Kim. 2018. Effects of Active Learning Environments and Instructional Methods in Computer Science Education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 934âĂŞ939. https://doi.org/10.1145/3159450.3159451

[9] Milo MK Martin, Daniel J Sorin, Bradford M Beckmann, Michael R Marty, Min Xu, Alaa R Alameldeen, Kevin E Moore, Mark D Hill, and David A Wood. 2005. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *ACM SIGARCH Computer Architecture News* 33, 4 (2005), 92–99.

[10] Jeffrey J McConnell. 1996. Active learning and its use in computer science. In *Proceedings of the 1st Conference on integrating Technology into Computer Science Education*. 52–54.

[11] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. 2006. Pair programming improves student retention, confidence, and program quality. *Commun. ACM* 49, 8 (2006), 90–95.

[12] David A Patterson and John L Hennessy. 2013. Computer Organization and Design MIPS Edition: The.

[13] Leo Porter, Cynthia Bailey Lee, Beth Simon, and Daniel Zingaro. 2011. Peer Instruction: Do Students Really Learn from Peer Discussion in Computing?. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. Association for Computing Machinery, New York, NY, USA, 45âĂŞ52. https://doi.org/10.1145/2016911.2016923

[14] Leo Porter, Saturnino Garcia, Hung-Wei Tseng, and Daniel Zingaro. 2013. Evaluating student understanding of core concepts in computer architecture. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 279–284.

[15] Beth Simon, Ruth Anderson, Crystal Hoyer, and Jonathan Su. 2004. Preliminary Experiences with a Tablet PC Based System to Support Active Learning in Computer Science Courses. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Leeds, United Kingdom) *(ITiCSE '04)*. Association for Computing Machinery, New York, NY, USA, 213âĂŞ217. https://doi.org/10.1145/1007996.1008053

[16] The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM)/IEEE Computer Society. 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.

[17] Ahmed Tlili, Fathi Essalmi, and Mohamed Jemni. 2016. Improving learning computer architecture through an educational mobile game. *Smart Learning Environments* 3, 1 (2016), 1–14.

[18] Dr Kenneth Vollmar and Dr Pete Sanderson. 2005. A MIPS assembly language simulator designed for education. *Journal of Computing Sciences in Colleges* 21, 1 (2005), 95–101.

[19] Kenneth Vollmar and Pete Sanderson. 2006. MARS: An Education-Oriented MIPS Assembly Language Simulator. *SIGCSE Bull.* 38, 1 (March 2006), 239âĂŞ243. https://doi.org/10.1145/1124706.1121415