Reducing the Carbon Footprint of EdTech with Repurposed Devices

Jennifer Switzer*, Subash Katel†, Jaemok Christian Lee‡,
Ashwin Rohit Alagiri Rajan §, Ryan Kastner ¶, and Pat Pannuto ‖
Computer Science and Engineering. UC San Diego
La Jolla, CA, USA
Email: *jswitze@ucsd.edu, †skatel@ucsd.edu, ‡jal105@ucsd.edu,
§aalagiri@ucsd.edu, ¶kastner@ucsd.edu, ‖ppannuto@ucsd.edu

***Abstract*—Education technology (EdTech) is an important tool for streamlining and improving course administration and teaching. Many modern EdTech tools rely on cloud services to host containerized applications. While this is convenient, it is also costly in terms of both dollars and carbon emissions. We propose the alternative approach of hosting containerized EdTech applications on local clusters of upcycled Android devices.**

We perform an evaluation of the Google Pixel Fold for handling educational workloads. Our findings suggest that such repurposed device could effectively bridge the gap between mobile and traditional computing platforms in education, open new avenues for accessible educational computing environments.

I. Introduction

Cloud computing has a significant carbon footprint, with the industry being responsible for 2.5-3.7% of worldwide greenhouse gas emissions [9]. While sustainable computing efforts have traditionally focused on energy efficiency, about 50% of the carbon emissions associated with datacenter hardware is incurred during manufacture [1]. This number—also referred to as computing's embodied carbon—is exacerbated by the fact that hardware is frequently replaced to maximize performance.

One way to curb embodied carbon is by reducing the demand for new machines. Discarded smartphones are a plentiful source of surprisingly powerful computing hardware. Since they are already manufactured and would otherwise be discarded, they incur no embodied carbon. By redeploying them, we prevent the manufacture of carbon-intensive servers. Previous work has shown that a small cluster of decommissioned smartphones can—at a significantly lower carbon footprint than traditional cloud computing—be repurposed to approximately match and sometimes even exceed the performance of a new, modern server using benchmarking suites with synthetics workloads [14].

In this work, we propose the repurposing of unwanted smartphones for a specific use-case: Education technology (EdTech).

Many recent EdTech tools make use of containerized applications, which in our experience are often deployed using cloud providers like Amazon Web Services (AWS). While this simplifies initial deployment, it is both expensive to maintain (in terms of the cost per hour to rent machines) and carbon-intense.

EdTech applications are an attractive target for repurposed hardware. First, they are generally less latency-sensitive than commercial applications. If a student has to wait an extra second or two for their grade, it is likely not a problem. Second, many EdTech applications exhibit extremely bursty behavior. Recent studies have shown that student submission patterns for online assignments are highly concentrated near deadlines. Nieberding and Heckler found that approximately 50% of students completed assignments within the final 24 hours before the deadline, with a steep increase in submissions in the hours immediately preceding the due date [10]. This pattern results in short periods of intense computational demand followed by longer periods of relative inactivity. Smartphones are a good fit for these bursty workloads due to their very low idle power. Unlike traditional servers that consume significant energy even when idle, smartphones can enter deep sleep states during periods of inactivity, dramatically reducing power consumption. When demand spikes, they can quickly wake and process the workload, then return to a low-power state [13]. This means that in addition to the embodied carbon savings possible by employing repurposed hardware, there is also the potential for a reduction in operational emissions.

The rest of this work outlines our design and implementation of a basic smartphone-based EdTech platform, and provides benchmarking results for two containerized EdTech applications, PrairieLearn and Jupyter notebook.

II. Background

Many existing EdTech workloads make use of containerized applications (e.g. Docker). Some prominent examples in computer science education are outlined below.

a) Gradescope Autograder: Gradescope is an educational platform that, among other features, allows students to submit assignments. Gradescope includes an autograder feature for programming assignments. This allows instructors to define a grading script that is automatically run against student submissions. In the background, the autograder is implemented using Docker containers. The instructor's grading script is built into a Docker image, and each time a student submits an assignment a new instance is spun up [4].

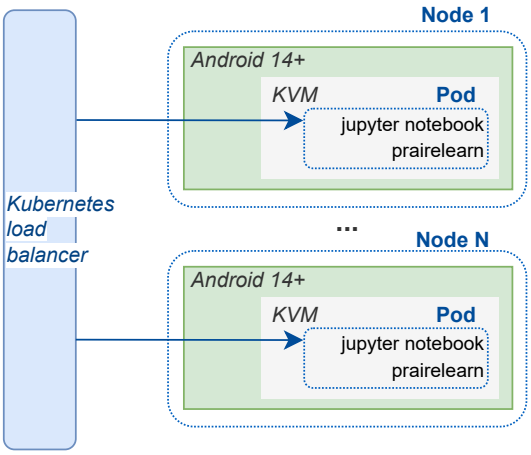

Fig. 1. A visual representation of the the cluster design. Phones running Android 14 and above are compatible with the system; We deploy a KVM instance on each device and connect the devices in a Kubernetes cluster. Each device acts as a Kubernetes node and runs the target EdTech container. We have experimented with both Jupyter notebooks and PrairieLearn. The Kubernetes load balancer could be run either on an external machine or another smartphone.

Depending on the implementation, the autograder containers might be run on local machines or in the cloud. In our experience, they are often run on AWS.

b) PraireLearn: PrairieLearn is a web-based homework platform. It allows instructors to define problem templates with variable parameters that can be automatically randomized to generate new questions [15]. PrairieLearn is typically run within Docker [12]. Similar to the Gradescope Autograder, a course's specific PrairieLearn instance might be hosted locally or with a cloud provider like AWS.

c) JupyterHub: JupyterHub allows a school or organization to run containerized Jupyter notebooks in the cloud or locally. It is built on Kubernetes using Docker [5].

III. System Design

While cloud providers like Google Cloud and AWS provide easy access to hardware, they are both expensive and carbon-intensive to operate [9]. The main design goal of our proposed system is to make local containerized deployments more accessible for EdTech applications.

We note that cloud-hosted services have two main advantages over local deployments:

1) No CapEX expenditure: Although the hourly price to rent cloud machines is significant, they require no up front investment.
2) Easier implementation: Deploying containers locally requires properly configured and maintained hardware. In many cases, it is easier to simply request cloud instances.

Assuming that we are able to source real unwanted hardware, our repurposed system is able to address the first problem. The value depreciation for Android smartphones is significant [6], making unwanted Androids a good target for building out a low-cost system.

Point #2 is difficult to address, however. Previous work has observed that implementing arbitrary software on top of the Android ecosystem is particularly difficult [7], [14]. Asking under-resourced instructors to work within such a difficult environment is a tall ask.

However, we find that recent versions of Android make this problem significantly more surmountable with the inclusion of virtualization capabilities in the Android kernel.

In Android 14 and above, KVM (kernel-based virtual machine) is built into the kernel. We use this capability to implement the system outlined in Figure 1. Each smartphone runs an Ubuntu-based virtual machine, on top of which we deploy a Kubernetes Pod. The Pod hosts the Docker container of interest–we test both Jupyter notebooks and PrairieLearn. A collection of smartphones can be connected into a Kubernetes cluster with a load balancer to distribute jobs.

We have built an Ubuntu image that can be deployed on any device running Android 14 and above. Setting up a new device requires only a handful of terminal commands, which can be bundled into a script. We plan to make the image and associated script open-source.

Our test system makes use of a number of Google Pixel Folds, which were released in 2023. We made the decision to focus on the Pixel Fold for practical reasons–our lab had a large surplus of these devices. While the Pixel Fold is quite modern, with the short 2-3 year average lifespan of consumer smartphones, we can reasonably expect that in the next 1-2 years smartphones with similar capabilities will become available on the secondary market.

IV. Benchmarking

This section provides an overview of the capabilities of the Pixel Fold smartphone in terms of general compute and GPU metrics. GPU capabilities in particular are interesting given growing student interest in AI/ML, as well as the significant dollar cost of purpose-built GPUs.

A. OpenCL Benchmarks

We use the mixbench OpenCL benchmark [8] to evaluate the GPU performance characteristics of the Google Pixel Fold. We compare against a reference NVIDIA GTX 1080 Ti, which are commonly used for coursework at our university.

The GFLOPS vs Arithmetic Intensity graph demonstrates that the Pixel Fold reaches a peak performance of about 700 GFLOPS for FP32 operations, while the 1080 Ti achieves approximately 10,000 GFLOPS (difference = 15.3x). For integer operations, the difference is more pronounced where the Fold reaches a maximum of 189 GIOPS while the 1080 Ti reaches around 4200 GIOPS (difference = 22x).

The Execution time vs Arithmetic Intensity is arguably the more important graph and we see similar result to the performance graph. The FP32 Execution time peaks at 26.60 ms for an Arithmetic Intensity of 128.250 for the Fold while

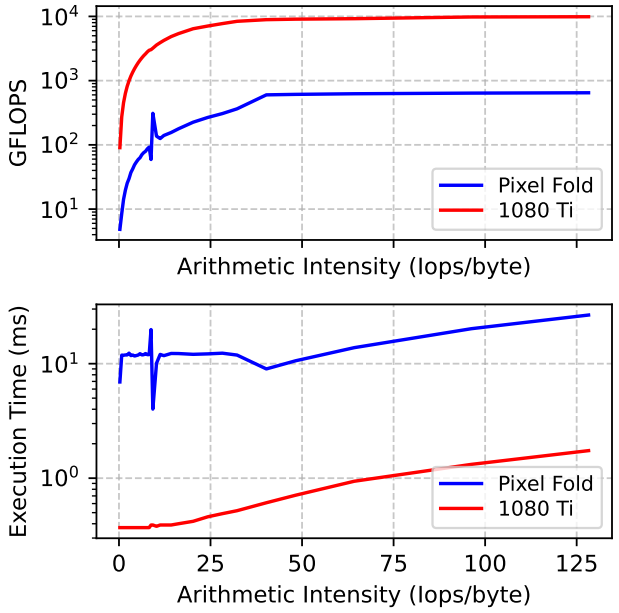

Fig. 2. Single Precision GPU characteristics for Pixel Fold and 1080 Ti

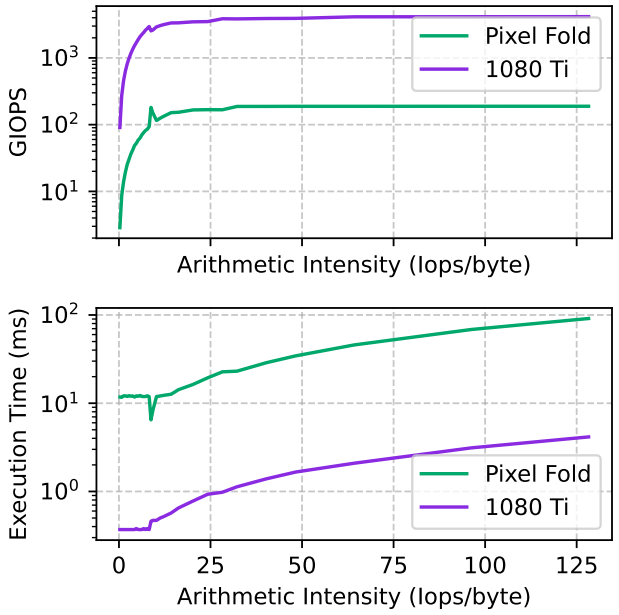

Fig. 3. Integer GPU characteristics for Pixel Fold and 1080 Ti

the 1080 Ti tops out at 1.74 ms (difference = 15.3x). The INT32 Execution times are much more pronounced where the Pixel Fold tops out at 91.19 ms and the 1080 Ti at around 4.15 ms (difference = 22x). The differences being the same as Performance is expected, but there are some noise values in the Fold's execution times particularly at 8.75 and 9.25 Iops/Byte and Flops/Byte Intensities.

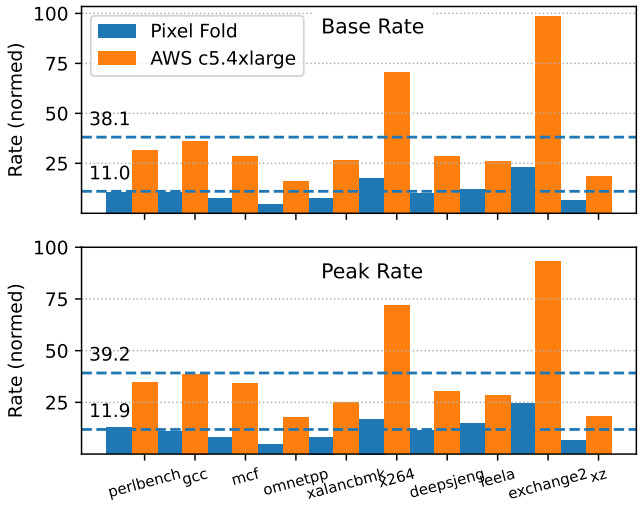

Fig. 4. Measure of peak intrate for the SPEC intrate suite for Pixel Fold versus a c5.4xlarge AWS instance.

B. SPEC Benchmarks

The SPEC CPU 2017 benchmarking tool was chosen to assess the processing capabilities of the Linux VM running on the Pixel Fold. The CPU performance was specifically targeted by running tests from the SPECrate Integer Test Suite. This test suite runs programs designed for CPU intensive operations such as filtering large quantities of spam (500.perlbench_r), or recursively generating non-trivial Sudoku puzzles (548.exchange2_r). As these programs are executed, different metrics are collected. Among these metrics is the base rate. The base rate is defined as:

$$\text{base rate} = \text{number of copies} * \frac{\text{time on reference machine}}{\text{time on system under test}} \quad (1)$$

where the number of copies is the number of concurrently running copies of the benchmark. Figure 1 shows a bar graph of the base rate of the integer rate test suite run on the Pixel Fold. We compare against an AWS EC2 instance of type c5.4xlarge, which costs 16.32 USD per day.

V. EdTech Applications

This section outlines the performance of the Google Pixel Fold on two types of EdTech applications, Jupyter notebooks and PrairieLearn.

A. Jupyter Server Hosting

We deploy and test two Jupyter notebooks: a basic educational notebook and a more resource-intensive machine learning notebook.

1. Machine Learning Notebook

This notebook contained code for training a convolutional neural network (CNN) on the MNIST dataset.

As shown in Figure 5, the ML notebook on the Pixel device maintained a consistent CPU utilization of around 60%

throughout the test. We also deployed this notebook on an n2d-standard-8 Google Cloud machine, which has 8 vCPUs and 32 GB RAM. The n2d machine exhibited a CPU utilization of around 50%. It completed the training task approximately 5x faster than the Pixel Fold. This is not too surprising given that the Pixel Fold has only 12 GB of memory compared to the n2d machine's 32 GB.

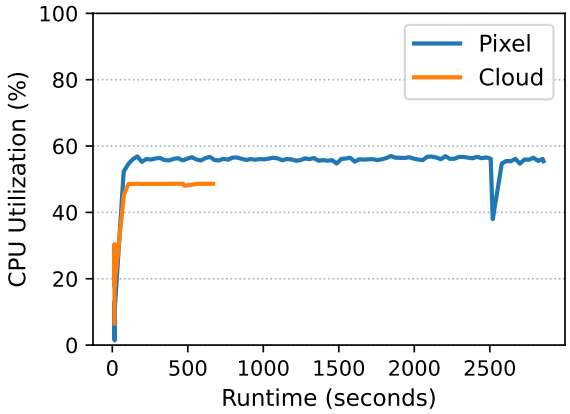

Fig. 5. System CPU Utilization for Machine Learning Notebook on Pixel vs Cloud (n2d-standard-8 Google Cloud instance [3])

2. Basic Data Science/CV Notebook

This notebook contained typical data analysis and computer vision tasks used in introductory computer science course.

As illustrated in Figure 6, the basic notebook on the Pixel device shows comparable latency compared to the n2d-standard-8 deployment. The CPU utilization is also comparable.

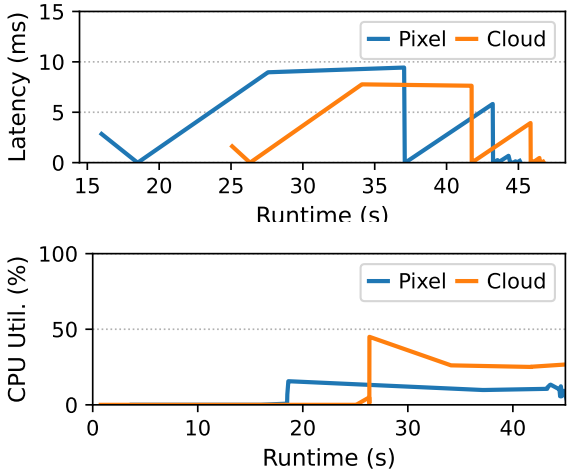

Fig. 6. Latency and CPU utilization comparison for Basic Data Science/CV Notebook on Pixel vs Cloud (n2d-standard-8 Google Cloud instance [3])

B. PrairieLearn

PrairieLearn is a software tool that allows for the administration of online courses. Users can create and manage

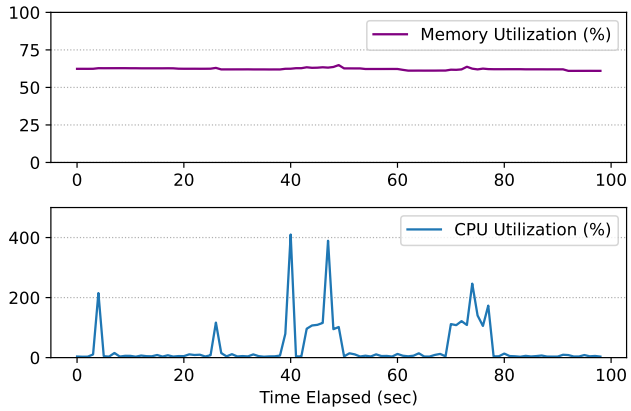

Fig. 7. Memory and CPU Utilization of an Exam Administered through PrairieLearn on Pixel Fold. CPU peaks occur when grading is initiated.

TABLE I
A SUMMARY OF EMBODIED AND OPERATIONAL EMISSIONS FOR ONE C5.4XLARGE INSTANCE VERSUS FOUR PIXEL FOLDS.

	Embodied	*Operational*		*3-year total*
		Power	3yr carbon	
c5.4xl	336 $kgCO_2$-e	144 W	252 $kgCO_2$-e	588 $kgCO_2$-e
Pixel	0 $kgCO_2$-e	80 W	140 $kgCO_2$-e	140 $kgCO_2$-e

classes through tasks such as assigning students autograded assignments. We host a sample PrairieLearn course on a Pixel Fold and measure the CPU and memory utilization for the scenario of a student completing an autograded exam. Questions ranged from basic addition, to matrix multiplication.

Figure 2 shows multiple distinct peaks in both memory and CPU usage following a student action of turning in an answer. While the phone's CPU was only periodically under stress, the relatively high memory utilization means that it would only be possible to host 1-2 instances of PrairieLearn per device.

C. Key takeaway

We find that the smartphone studied is capable of hosting the EdTech applications of interest with reasonable performance. A limiting factor is the relatively low memory-to-CPU ratio typical of the smartphone platform. The device studied–the Pixel Fold–has more memory than most smartphones with 12 GB for its 8 cores. However, this is still significantly less than the 32 GB provisioned for the 8 core n2d Google Cloud instance. Depending on the target application, this may or may not be a problem.

For simple applications, the performance seems to be more than sufficient.

VI. CARBON SAVINGS

We hypothesize that a smartphone-based EdTech system will reduce carbon emissions compared to a traditional cloud-computing based alternative. There are two ways that the smartphone system saves carbon: First, by reducing embodied emissions by preventing the manufacture of new, purpose-built

devices; and second, by reducing operational emissions due to the relatively power-efficient operation of the smartphone platform.

We compare against the alternative of using Amazon web services (AWS) for docker container hosting. We make this decision for two reasons. First, in our experience AWS is commonly used for EdTech applications. Second, a third-party tool exists for estimating the carbon footprint of AWS instances [2].

We compare the Pixel Fold against an AWS c5.4xlarge EC2 machine, for which we have collected SPEC benchmark data. Based on SPEC results, a c5.4xlarge EC2 machine is able to achieve approximately 4x the throughput of a Pixel Fold on the SPEC intrate benchmark suite. This is shown by the dotted horizontal lines in Figure 4.

We therefore assume that it takes approximately 4 Pixel Fold smartphones to match the computational capabilities of a c5.4xlarge machine. We compare the three-year carbon emissions of a single c5.4xlarge instance against a cluster of four Pixel Fold smartphones.

a) Embodied emissions: Embodied emissions are the emissions associated with the manufacture of the device. In the case of our smartphone-based system, we assume that the phones in use are repurposed and therefore incur no embodied emissions. For a full-scale system it would be necessary to take into account the embodied footprint of peripheral hardware like USB hubs, but for this analysis we only consider the hardware itself.

For the c5.4xlarge EC2 instance, we use a third-party calculator to estimate the embodied carbon of the instance as 336 kgCO2 [2].

b) Operational emissions: To get a rough estimate of operational emissions, we assume that all devices are operating at 100% utilization. This is generally the case when running the SPEC benchmark suite.

We use the same calculator as above [2] to estimate the power consumption of the AWS c5.4xlarge EC2 instance at 100% utilization as 144 W. We estimate the Pixel Fold's peak power consumption to be no more than 20 W. Since there are four phones in our comparison cluster, the total power draw of the phone system is 80 W at maximum.

We assume that we are operating within the California grid with an average grid carbon intensity of 0.200 kgCO2/kWh [11].

The results of the analysis are summarized in Table I. The phone-based system incurs 4x fewer emissions.

REFERENCES

[1] Andreas Busa. Life cycle assesssment of dell r740 server. https://www.delltechnologies.com/asset/en-us/products/servers/technical-support/Full_LCA_Dell_R740.pdf, 2019. Accessed: 2021-06-01.

[2] Teads Engineering. Estimator for aws instances. https://engineering.teads.com/sustainability/carbon-footprint-estimator-for-aws-instances/, 2024.

[3] Google. Google cloud n2d machine types. https://cloud.google.com/compute/docs/general-purpose-machines#n2d_machines, 2024.

[4] Gradescope. Gradescope autograder documentation: Technical details. https://gradescope-autograders.readthedocs.io/en/latest/specs/. Accessed: 2024-08-05.

[5] Project Jupyter. Jupyterhub. https://jupyter.org/hub. Accessed: 2024-08-05.

[6] Adrian Kingsley-Hughes. Your android smartphone could be worthless after a few years, 2022. Accessed: 2024-08-04.

[7] Noah Klugman, Veronica Jacome, Meghan Clark, Matthew Podolsky, Pat Pannuto, Neal Jackson, Aley Soud Nassor, Catherine Wolfram, Duncan Callaway, Jay Taneja, and Prabal Dutta. Experience: Android resists liberation from its primary use case. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 545–556, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Elias Konstantinidis and Yiannis Cotronis. A quantitative roofline model for gpu kernel performance estimation using micro-benchmarks and hardware metric profiling. *Journal of Parallel and Distributed Computing*, 107:37–56, 2017.

[9] Hessam Lavi. Measuring greenhouse gas emissions in data centres: the environmental impact of cloud computing. https://www.climatiq.io/blog/measure-greenhouse-gas-emissions-carbon-data-centres-cloud-computing. Accessed: 2024-08-04.

[10] Megan Nieberding and Andrew F. Heckler. Patterns in assignment submission times: Procrastination, gender, grades, and grade components. *Phys. Rev. Phys. Educ. Res.*, 17:013106, Jun 2021.

[11] California Independent System Operator. California iso. https://www.caiso.com/. Accessed: 2024-10-02.

[12] PrairieLearn. Prairielearn. https://prairielearn.readthedocs.io/en/latest/. Accessed: 2024-08-05.

[13] Pijush Kanti Dutta Pramanik, Nilanjan Sinhababu, Bulbul Mukherjee, Sanjeevikumar Padmanaban, Aranyak Maity, Bijoy Kumar Upadhyaya, Jens Bo Holm-Nielsen, and Prasenjit Choudhury. Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage. *ieee Access*, 7:182113–182172, 2019.

[14] Jennifer Switzer, Gabriel Marcano, Ryan Kastner, and Pat Pannuto. Junkyard computing: Repurposing discarded smartphones to minimize carbon. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 400–412, New York, NY, USA, 2023. Association for Computing Machinery.

[15] Matthew West, Geoffrey L Herman, and Craig Zilles. Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. In *2015 ASEE Annual Conference & Exposition*, pages 26–1238, 2015.