Bringing Networked Sensing to High School

Devin Jean
devin.c.jean@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Gordon Stein
gordon.stein@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Saman R. Kittani
saman.r.kittani@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Ákos Lédeczi
akos.ledeczi@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Abstract

Many block-based programming environments successfully engage novice programmers but offer limited access to the physical world and its networked systems. This paper presents a visual programming environment designed to make both networked sensing systems and IoT concepts accessible to secondary students. Building upon existing block-based programming paradigms, this environment introduces intuitive distributed computing abstractions that enable students to create networked sensing systems while maintaining the accessibility of block-based programming. Additionally, the paper presents a pathway for students to transition from blocks to Python. The paper describes the system architecture, computing abstractions, related tools, and empirical studies demonstrating its effectiveness in teaching foundational IoT concepts in secondary education.

CCS Concepts

• **Applied computing** → **Interactive learning environments**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Social and professional topics** → **K-12 education**.

Keywords

IoT, Secondary Education, Block-based Programming, Distributed Computing

1 Introduction

The proliferation of Internet of Things (IoT) devices and networked sensing systems has transformed how we collect, process, and act upon data from our physical environment. From smart home sensors to environmental monitoring networks, these distributed systems now form the backbone of modern data-driven applications. As IoT and networked sensing become increasingly prevalent, we need to prepare students early - starting in K-12 education - to understand and work with these technologies [9, 11]. This paper presents an ecosystem of educational tools that has been shown to be effective in teaching foundational networked sensing and IoT concepts in secondary education.

NetsBlox, a visual programming environment, and its associated tools were designed specifically to introduce advanced computing concepts to novices [2]. Distributed computing, networking, IoT, robotics and machine learning are in everyday use yet are typically only taught to computer science majors in college. The platform builds on familiar block-based programming concepts, making it accessible to teachers with experience in tools like Scratch [8] or Snap*!* [3], while extending far beyond these tools by supporting advanced computing in an intuitive manner.

The architecture of the NetsBlox ecosystem is depicted in Figure 1. The heart of the system is the NetsBlox Cloud Server that provides several services, manages projects, routes messages and orchestrates collaborative editing. All clients connect to the server through a well-documented RESTful API. The server provides access to a large number of curated online data sources and services such as Google Maps or USGS earthquake data. The main client is the browser-based visual programming environment. Users who are ready to switch to Python can use a custom IDE specifically designed to support the block-to-text transition called PyBlox. Other components include PhoneIoT, a mobile app that connects mobile phones to NetsBlox as programmable IoT sensors, and RoboScape, a robotics platform. Virtual robots and sensors are supported through RoboScape Online and IoTScape. Finally, the NetsBlox Virtual Machine (VM) makes it possible to run NetsBlox projects on various embedded platforms.

To overcome resource constraints, PhoneIoT leverages devices students already own - their smartphones - as IoT sensors and actuators, eliminating the need for expensive specialized hardware. RoboScape allows schools to start with free virtual environments and gradually add physical robots as resources permit. What sets

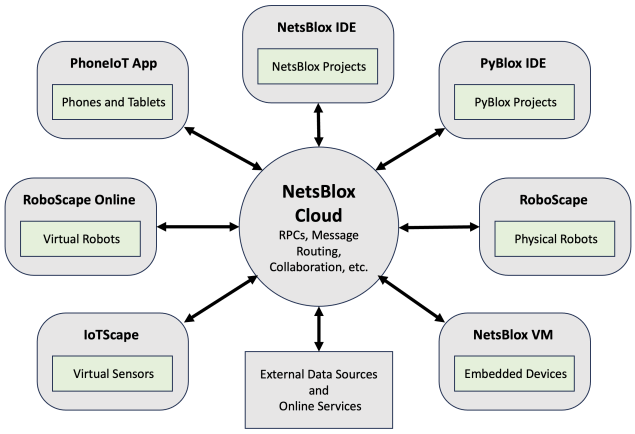

Figure 1: Overview of the NetsBlox ecosystem

NetsBlox apart from other educational tools like Scratch, Snap!, AppInventor [10], and Microsoft MakeCode [1] is that it uses just two unifying distributed computing abstractions to provide much broader support for teaching fundamental IoT concepts, including networking, distributed computing, remote sensing and actuation, in an accessible manner.

2 Programming Abstractions

NetsBlox extends block-based programming environments to make distributed computing concepts accessible to novice programmers. Built on Snap!'s foundation, NetsBlox removes the traditional walls around educational programming environments by enabling students to create applications that can access online resources and communicate with each other and additional devices over the internet. This approach lets students build engaging projects like multiplayer games, interactive maps with real-time data, remote-controlled robots, and IoT applications while learning fundamental concepts in distributed computing.

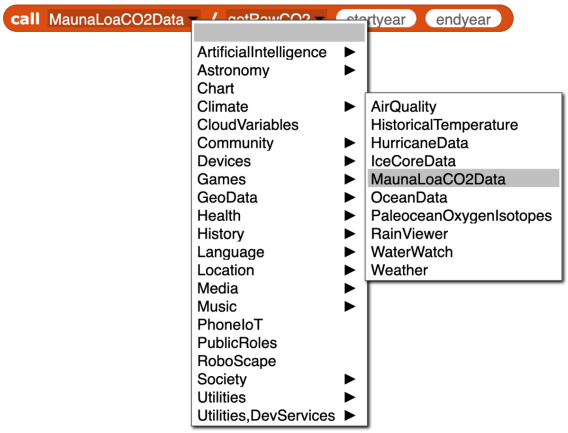

Figure 2: Related RPCs are grouped into services. The first menu of the call block lets users select the desired service which in turn reconfigures the second menu, the available RPCs. Selecting an RPC, reconfigures the input argument slots. Climate sub-menu is shown.

Remote Procedure Calls (RPCs) form one of NetsBlox's core abstractions, providing a simple yet powerful way to access online data and services. Related RPCs are grouped into *Services* (Figure 2). Through a single "call" block that dynamically configures itself based on the selected service and function, students can access everything from weather data to Google Maps to movie databases without dealing with complex APIs, parsing or authentication. This abstraction shields students from the technical complexities of web services while preserving the educational value of working with real-world data and online services. The RPC mechanism is particularly elegant because it mirrors the familiar concept of custom blocks (i.e., functions) - both have inputs, outputs, and wait for results - making it intuitive for students who already understand basic programming concepts.

Message passing represents NetsBlox's second fundamental abstraction, enabling direct communication between running programs. While similar to events in traditional block-based environments, messages in NetsBlox can carry data and travel between any computers connected to the internet. This abstraction allows students to create distributed applications like chat rooms, multiplayer games, and collaborative tools. Messages have types defined by a name and a set of input fields, requiring students to think about protocols and APIs - key concepts in distributed computing. The system supports both local addressing within a project and global addressing across the internet, introducing students to important networking concepts while keeping the implementation details manageable.

3 Networking

The NetsBlox message passing abstraction allows a lot of flexibility to teach various networking concepts. The implementation of message passing is centralized: all messages go through the server. But from the user's perspective, it appears to be peer to peer. The "send msg" block requires the user to provide the address of the recipient. A running project can be uniquely identified by the (username, project name) pair, as each of these must be unique in its own context. If we want to send a message to the project "*TempSensing*" of the user "Hacker1" we would use the address: "*TempSensing@Hacker1*". If the user is running the given project at the time, they would receive the message.

NetsBlox allows users to define custom message types with specific names and data fields. The dropdown menu in the send message block allows the user to pick a message type, which in turn reconfigures the block to show the available data slots. The "when I receive" event handler block works in a similar fashion: selecting the message type shows the available data slots as (local) variables that can be used in the corresponding handler script. This feature mirrors the concept of protocols in network communication, where messages are structured with defined headers and payloads for specific purposes. By designing message types, students learn the importance of standardized message formats for effective communication between different parts of a distributed system.

4 Sensing and Actuation

Teaching the Internet of Things (IoT) requires a lab with parts and tools and a teacher with extensive experience. But most high school students already own a device with a rich set of sensors and out-of-the-box internet connectivity. PhoneIoT is an open-source mobile application for Android and iOS devices that turns smartphones and tablets into IoT devices [4]. PhoneIoT leverages the RPC and message passing to interact with mobile devices through ordinary NetsBlox code. When the PhoneIoT app is launched, it automatically connects to the NetsBlox cloud server and exposes itself under a hidden ID and password shown in the app's menu. These login credentials can then be used to access the device remotely, either via RPCs or message passing. In terms of sensing, PhoneIoT supports two access paradigms: polling and streaming. The polling approach allows students to call an RPC (one for each sensor) to immediately receive the current value from the device. For instance, the PhoneIoT "getAccelerometer" RPC returns the current accelerometer value (as

a list of three numbers), the “getLocation” RPC returns the current GPS location (as a latitude/longitude pair), the “getPressure” RPC returns the current air pressure, and so on.

These sensor polling RPCs in PhoneIoT are quite convenient for accessing a one-off value for initialization or otherwise infrequent use, but for continuous access it can be tedious and inefficient. For instance, while it is easy enough to listen to one sensor by polling in a loop, listening to multiple sensors in this way can be cumbersome to code, and requires a round trip exchange with the phone over the network for each and every sensor request. To solve this issue, PhoneIoT’s second sensor access method is “streaming.” With this approach, a single RPC is invoked to request a set of sensors send updates at specific update intervals. This eliminates the round trip, and allows the phone to bundle multiple sensor values together into a single sensor update packet, where appropriate. To use streaming, students can call the “listenToSensors” RPC and provide it the desired list of sensor/update interval pairs. These updates can then be received as special documented message types. For instance, the accelerometer can be streamed by receiving messages of type “accelerometer” with fields “x,” “y,” and “z.”

As an example of PhoneIoT sensing in practice, Figure 3 contains all the code that is needed to create a simple single-axis acceleration plotter using the streaming access method. Figure 4 demonstrates the visualization that this code produces. With only slightly more complexity, a full 3-axis plotter could be created. Or, rather than simply plotting these values, they could be used as input to control some other program logic. For instance, the phone can be turned into a remote game controller.

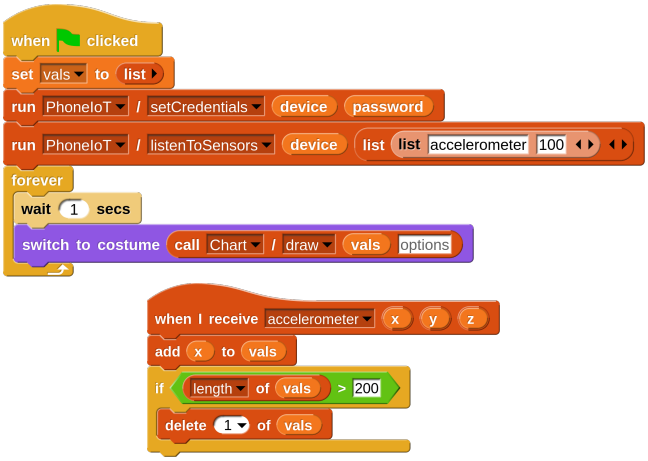

Figure 3: Single-axis acceleration plotter (code). Note that the "run" block works the same as the "call" block, but it disregards the return value.

PhoneIoT offers actuation through a customizable graphical display on the phone screen. Using standard NetsBlox RPCs, students can populate this display with various widgets and manipulate them in real-time. For example, students can create an image display widget and populate it with Google Maps imagery by combining the existing GoogleMaps NetsBlox service with PhoneIoT’s ’getLocation’ RPC. PhoneIoT widgets extend beyond simple display

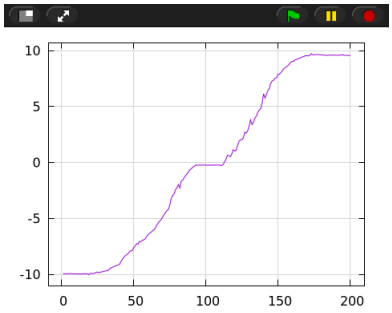

Figure 4: Single-axis acceleration plot. In this 20-second experiment, the phone lay flat at first, then was set on its edge slowly, and finally, turned face down.

functionality by offering interactive features. Students can enable interactivity by registering an ’event’ when creating a widget - this defines a custom message type that the device sends back to the project whenever a user interacts with the widget.

4.1 Robotics: Physical and Virtual

NetsBlox’s support for robots called RoboScape takes a fundamentally different approach to educational robotics by having student code running in the browser rather than on the robots themselves [7]. Students write programs in NetsBlox to control the robots by sending commands over the network. This networked architecture means students don’t need to learn any special robotics programming concepts - they simply use the same Remote Procedure Calls (RPCs) and message passing features they’ve already learned in NetsBlox.

RoboScape Online extends the networked architecture of RoboScape into a browser-based 3D simulation environment while maintaining complete compatibility with physical robots [12]. Since both virtual and physical robots use the same network protocol and RPC interface, student code works identically whether controlling simulated or real robots. However, the simulation environment enables capabilities that would be prohibitive with physical hardware, such as LIDAR sensors or precise position tracking.

In RoboScape, the robots expose a set of sensors to NetsBlox users. A program can send a “get range” command to get the distance detected by the robot’s ultrasonic rangefinder. While this polling approach is suitable for most sensors, others may require an event-based approach to help enable an interrupt-like behavior in the control program. For example, the robot’s front-mounted tactile “whisker” sensor is used to detect collisions. By subscribing to events through the “listen” RPC, a program can receive events when the robot’s state changes, including if the whisker sensors are touching something. Figure 5 demonstrates the latter approach. The programs command a robot to drive forward and then waits for an event. Providing sensors that work with both approaches helps familiarize students with multiple modalities of interacting with sensors.

RoboScape Online’s simulation environment exposes its virtual sensors to users in similar ways, with each sensor type presented as its own web service with RPCs specific to its capabilities. These

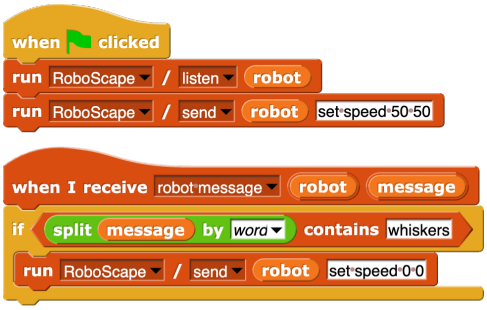

Figure 5: Example "safe" driving program using whisker sensor events

services are created through the IoTScape platform, which allows arbitrary Internet-connected devices to be exposed to the NetsBlox ecosystem through UDP message exchange with the NetsBlox server.

To illustrate how NetsBlox enables high school students to create truly distributed sensing and actuation systems consider Figure 6. This simple NetsBlox project runs on three devices: a computer with a web browser, a phone and a robot (real or simulated). It uses a single PhoneIoT joystick widget to control a RoboScape robot.

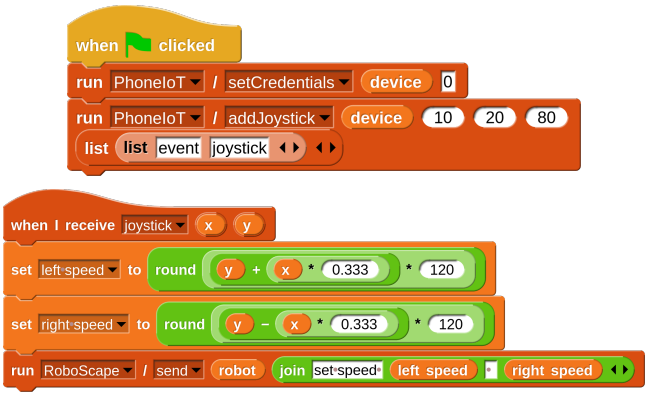

Figure 6: PhoneIoT joystick-based robot controller

4.2 Virtual Machine

NetsBlox VM represents a significant advancement in making distributed and embedded programming accessible to students. At its core, it's a virtual machine that allows block-based programs to run on practically any device while preserving all the sophisticated programming features that make NetsBlox powerful. Written in Rust for its emphasis on safety and reliability, the VM includes a plugin system that lets developers customize how different programming blocks behave, opening up countless possibilities for educational applications [6].

NetsBlox32 uses **NetsBlox VM** to bring the power of NetsBlox to affordable ESP32 microcontrollers. What makes NetsBlox32 particularly innovative is its completely wireless approach to programming. Students can write their code in the familiar NetsBlox editor and wirelessly upload it to their device, eliminating the usual hassle of physical connections and complicated setup procedures. When students encounter problems, NetsBlox32 provides detailed visual feedback about errors, displaying visual stack traces attached directly to their block-based code to help them understand what went wrong.

Using NetsBlox32 ESP32 microcontrollers can easily be connected to any number of off-the-shelf components. One simply needs to connect the component to the microcontroller and update a simple JSON file to add a new entry for the peripheral. This JSON file can be remotely edited from the browser. Once this is done, simply power cycling the device will update its peripherals info and give students access to the new peripheral via a collection of "system call" or "syscall" blocks, which are simply the hardware equivalent of the familiar NetsBlox "call" block.

As an example, we have configured a NetsBlox32 device to have only a single peripheral: a BMP388 environmental sensor; the device is shown in Figure 7 (note that the large black box is a battery, in fact a rather excessive one for our needs). As for the code, Figure 8 provides all that is required to turn this device into a remote weather station. This project collects readings of pressure and temperature at 10Hz, and every 60 seconds broadcasts a message that causes the second script to upload the data to the cloud. Running this as a separate script allows the upload to be performed in the background so as not to interrupt the data collection. Students can place this device somewhere secure, such as the roof of the school, and remotely access the stored data from the cloud when needed for plotting and analysis.

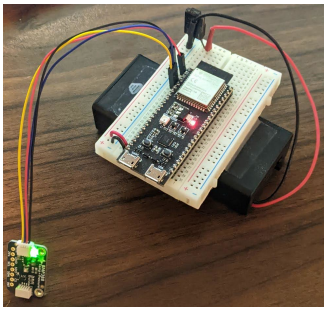

Figure 7: NetsBlox32 device with a connected environmental sensor (bottom left)

5 Beyond Blocks

At this point, we have seen that the simple distributed computing primitives provided by NetsBlox (i.e., RPCs and message passing) combined with the use of intuitive drag-and-drop program construction allow students to quickly learn the basics of several real-world networked computing topics, including robotics via RoboScape and the Internet of Things via PhoneIoT. However, despite offering all of these powerful programming capabilities, NetsBlox is fundamentally an educational platform, and as such, students wishing to delve deeper into networked computing will eventually have to migrate to contemporary textual programming environments.

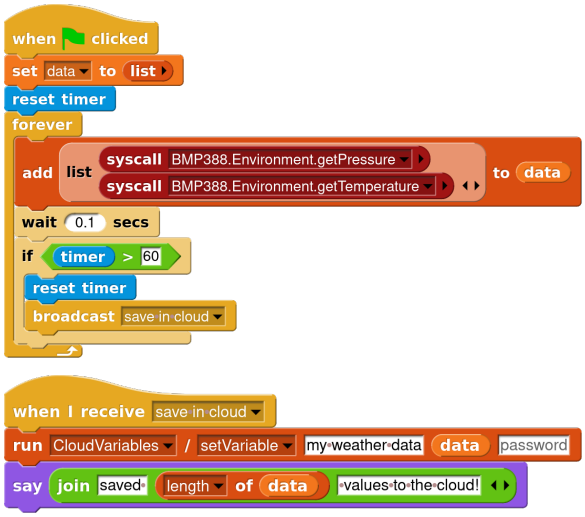

Figure 8: NetsBlox32 remote weather station code

```
vals = []
device = '9c9a19f6c11e'
password = '60ed2081'

@onstart()
def my_onstart(self):
    nb.phone_iot.set_credentials(self.device, self.password)
    nb.phone_iot.listen_to_sensors(self.device, {
        'accelerometer': 100,
    })
    while True:
        time.sleep(1)
        self.costume = nb.chart.draw(self.vals)

@nb.on_message('accelerometer')
def my_onmessage(self, x, y, z):
    self.vals.append(x)
    if len(self.vals) > 200:
        del self.vals[0]
```

Figure 9: A PyBlox implementation of the single-axis acceleration plotter shown in Figure 3.

For this reason, we have developed PyBlox, a transitional platform that bridges the gap between block-based programming in NetsBlox and text-based programming in Python. With PyBlox, students write (exclusively) Python code, but are provided with high-level Python APIs for the vast majority of NetsBlox features, including NetsBlox RPCs, message passing, and even (pseudo-)parallel event-based computing. As a result, students are able to continue using both RoboScape and PhoneIoT from Python. Figure 9 presents a short example program which connects to PhoneIoT from Python, making use of the boilerplate-free turtle graphics capabilities provided by the dedicated PyBlox code editor.

Importantly, PyBlox, both the code editor and the programs students construct using it, run as standard Python programs on the host platform. Because of this, PyBlox grants students the full, unlimited power of Python. However, if and when students choose to move on from PyBlox to "plain" Python (i.e., simply without using the PyBlox editor), all of the programming features of PyBlox remain accessible through the underlying `netsblox` library used by PyBlox. For example, the `nb` object used to access NetsBlox networking features (as seen in Figure 9) can be constructed in "plain" Python as simply `nb = netsblox.Client()`. An example of using this plain Python package to access PhoneIoT is presented in Figure 10; note that `nb.wait_till_disconnect()` is only required to keep the program running and able to receive messages after reaching what would otherwise be the end of the program. With this, students are provided with a complete learning pathway from block-based programming in NetsBlox to text-based programming in plain Python, all while maintaining the advanced networked computing features of NetsBlox.

```
import netsblox
nb = netsblox.Client()

@nb.on_message('accelerometer')
def accel_handler(x, y, z):
    print(f'accel: {x} {y} {z}')

device = input('device id: ')
password = input('password: ')
nb.phone_iot.set_credentials(device, password)
nb.phone_iot.listen_to_sensors(device, {
    'accelerometer': 100,
})

nb.wait_till_disconnect()
```

Figure 10: A pure Python program which accesses a PhoneIoT device through NetsBlox.

6 Studies

We have conducted multiple classroom studies using various curricula focusing on different aspects of networked sensing and IoT that NetsBlox supports. These included one-week summer camps and multi-week after-school programs where students met weekly for 1-2 hours. Since we could not assume programming background, all programs began with general programming introduction.

Two clear conclusions emerged from these studies. First, even students with no prior programming experience were able to successfully create distributed applications, including accessing phone sensors and controlling robots, by the end of our short programs. Second, pre- and post-surveys and interviews consistently showed high student engagement and increased interest in computing.

The RoboScape Online platform was evaluated through two studies with high school students. A 2022 study [12] with 27 students and a 2023 study [13] with 26 sophomores evaluated platform effectiveness through surveys, interviews, and programming assessments. The 2023 study demonstrated significant positive changes in students' attitudes toward CS and robotics, particularly increased confidence in creative expression through computing. Student interviews revealed that both novice and experienced learners found the platform engaging, with custom environment creation and LIDAR-based activities proving most popular. Despite some reported technical issues with the NetsBlox interface, students found the robotics platform accessible and motivating.

The PhoneIoT platform evaluation through a summer camp [5] showed students found the material highly engaging, ranking PhoneIoT-specific projects as most appealing. Students particularly

appreciated the hands-on "real world" applications that PhoneIoT enabled.

While these studies involved self-selected STEM-interested students, potentially limiting generalizability, results consistently demonstrate that our platforms effectively engage high school students in networked sensing education while developing computational thinking skills and creative confidence.

7 Conclusions

Educational technologies that effectively support learning must balance accessibility with sophistication. This principle, captured in the metaphor of "low floor, high ceiling," emphasizes that learning environments should minimize barriers to entry while supporting progression toward advanced applications. Mitchel Resnick extended this framework by adding "wide walls," arguing that environments should span diverse domains to engage students with varying interests and backgrounds.

NetsBlox realizes and extends these principles through innovative distributed programming abstractions. By introducing Remote Procedure Calls and message passing primitives, NetsBlox maintains the accessibility of block-based programming while enabling students to create sophisticated distributed applications. The integration of physical and virtual components extends those capabilities into networked sensing and IoT systems. Through PhoneIoT, students transform smartphones into sensing and control devices, while RoboScape Online provides access to sophisticated virtual sensors like LIDAR and GPS that would be prohibitively expensive in physical form. These tools make advanced concepts in networking, sensing, and actuation accessible through engaging projects.

The platform's progression to Python through PyBlox further extends its educational reach. By providing the same distributed computing abstractions in Python, the ecosystem supports a seamless transition from visual programming to text-based coding, making it suitable for introductory college courses in IoT and distributed systems. Students can leverage the same PhoneIoT and RoboScape capabilities they learned in high school, but now within a professional programming environment.

Our empirical studies demonstrate that high school students can successfully grasp these advanced computing concepts when presented through NetsBlox's intuitive abstractions, showing significantly increased engagement with and understanding of networked sensing systems. The consistent positive feedback across multiple implementations validates our approach of using familiar programming paradigms to introduce complex distributed computing concepts.

This work contributes to computing education in several key ways. First, it demonstrates that sophisticated networking and IoT concepts can be made accessible to secondary students without sacrificing technical depth. Second, it provides a complete ecosystem of tools that work together seamlessly, addressing the fragmentation often found in educational technology stacks. Third, it offers a clear progression pathway from visual to text-based programming, addressing a critical transition point in computing education. Finally, by leveraging devices students already own and providing free virtual alternatives, it addresses equity concerns that often limit access to advanced computing education.

Looking forward, several opportunities exist for extending this work. We plan to expand the curriculum materials and teacher professional development resources to support broader adoption. The addition of machine learning capabilities to the NetsBlox ecosystem is under development, which will enable students to create intelligent IoT applications. We are also exploring partnerships with school districts to conduct longitudinal studies examining the long-term impact of early exposure to distributed computing concepts on students' computational thinking skills and career trajectories.

As networked sensing and IoT continue to transform our world, preparing students to understand and shape these technologies becomes increasingly critical. The NetsBlox ecosystem demonstrates that with thoughtful abstraction design and comprehensive tool support, we can empower the next generation to not just consume but create the networked systems of the future. By maintaining our commitment to accessibility while pushing the boundaries of what's possible in K-12 computing education, we hope to inspire a new generation of innovators who will leverage distributed computing to address the challenges of tomorrow.

Our empirical studies demonstrate that high school students can successfully grasp these advanced computing concepts when presented through NetsBlox's intuitive abstractions, showing significantly increased engagement with and understanding of networked sensing systems.

References

[1] Ball, T., Chatra, A., de Halleux, P., Hodges, S., Moskal, M., and Russell, J. Microsoft MakeCode: embedded programming for education, in blocks and TypeScript. In *Proceedings of the 2019 ACM SIGPLAN symposium on SPLASH-E* (2019), pp. 7–12.

[2] Broll, B., Lédeczi, A., et al. A visual programming environment for learning distributed programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Mar 2017), ACM, p. 81–86.

[3] Harvey, B., Garcia, D. D., Barnes, T., Titterton, N., Miller, O., Armendariz, D., McKinsey, J., Machardy, Z., Lemon, E., Morris, S., and Paley, J. Snap! (build your own blocks). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2014), SIGCSE '14, ACM, pp. 749–749.

[4] Jean, D., Broll, B., Stein, G., and Lédeczi, Á. Utilizing smartphones for approachable iot education in k-12. *Sensors 22*, 24 (2022), 9778.

[5] Jean, D., Grover, S., Ledeczi, A., and Broll, B. Phoneiot for teaching 'internet of things': Smartphones to promote accessible, engaging, and authentic experiences. In *Proceedings of the 17th International Conference of the Learning Sciences-ICLS 2023, pp. 2181-2184* (2023), International Society of the Learning Sciences.

[6] Jean, D., Stein, G., Broll, B., and Ledeczi, A. Unleashing creativity with wireless embedded programming for next-generation makers. In *EDULEARN24 Proceedings* (2024), IATED, pp. 2100–2108.

[7] Lédeczi, Á., Maróti, M., Zare, H., Yett, B., Hutchins, N., Broll, B., Völgyesi, P., Smith, M. B., Darrah, T., Metelko, M., et al. Teaching cybersecurity with networked robots. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (2019), pp. 885–891.

[8] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. Scratch: a sneak preview [education]. In *Creating, connecting and collaborating through computing, 2004. Proceedings. Second International Conference on* (2004), IEEE, pp. 104–109.

[9] National Academies of Sciences Engineering and Medicine. *A 21st Century Cyber-Physical Systems Education.* NATIONAL ACADEMIES PRESS, 2016.

[10] Patton, E. W., Tissenbaum, M., and Harunani, F. MIT app inventor: Objectives, design, and development. *Computational thinking education* (2019), 31–49.

[11] Stankovic, J. A., Sturges, J. W., and Eisenberg, J. A 21st century cyber-physical systems education. *Computer 50*, 12 (2017), 82–85.

[12] Stein, G., Jean, D., Brady, C., and Lédeczi, Á. Browser-based simulation for novice-friendly classroom robotics. *Frontiers in Computer Science 4* (2023), 1031572.

[13] Stein, G., Jean, D., Kittani, S., Deweese, M., and Lédeczi, Á. A novice-friendly and accessible networked educational robotics simulation platform. *Education Sciences 15*, 2 (2025), 198.