Volume 5 | Spring 2024

IEEE Student Magazine

Works from Students at the University of Cincinnati

IEEE
University of Cincinnati Branch

Contents

Foreword from the IEEE Student Magazine Editor-in-Chief

Natalia Lui, *IEEE Student Magazine, Editor-in-Chief*

I joined IEEE in 2021 as the project manager of the IEEE Student Magazine, witnessing firsthand the potential of an accessible published work for undergraduate students. The first publication of this magazine occurred right before COVID, reaching a standstill in development once it had lost its original founders. This project began from a desire to share the undergraduate electrical engineering and computer science stories hidden within each accomplished student. To promote this desire, I have worked tirelessly to ensure any STEM student from the University of Cincinnati can submit their articles to be peer reviewed, edited, and published as a professional testament to their academic careers.

The IEEE Student Branch at the University of Cincinnati is an enduring pillar of professionalism, and I cannot be more grateful to have worked with its brilliant executive team to put together this year's magazine. I am grateful to the students that wrote and edited these articles into a genuine testament of the capabilities within our student body. This magazine could not have been done without them. I hope this magazine will be a powerful demonstration of the students' achievements and growth throughout the years.

I am deeply grateful to the students who have poured their hearts and souls into making this magazine a reality. Their hard work and dedication have not gone unnoticed, and I am continually impressed by their professionalism and creativity. As we eagerly anticipate the release of each new issue, I am reminded of the boundless potential that lies within each student. I extend my heartfelt thanks to all who have contributed to this publication, whether through writing, editing, or support. Your contributions have enriched our community and strengthened our connections both within and beyond the university. I am honored to be part of this journey and look forward to witnessing the success of the IEEE Student Magazine at the University of Cincinnati.

Foreword from the IEEE Student Magazine Project Manager

Mettika Ukey, *IEEE Student Magazine, Project Manager*

My experience as the project manager for the IEEE Student magazine has been an exceedingly fulfilling journey. Seeing the magazine blossom from its inception into a final, tangible product has been incredibly rewarding. It has always been my goal to inspire readers by telling the stories of students at the University of Cincinnati, just like you and me. Each volume represents the hard work, passion, and talent of our robust student body. I cannot wait for this magazine to be shared with hundreds within our community, sparking ideas and imagination along the way.

It takes a village to raise a child, and this has been no different. I would like to thank everybody at the University of Cincinnati chapter of IEEE for their unwavering support. Their contributions, whether big or small, have been essential in bringing this magazine to life. I would also like to extend my heartfelt gratitude to Natalia for her invaluable guidance. She has been amazing mentor and an incredible friend. As the torch of editor-in-chief gets passed down to me, I will work hard to make sure the magazine remains a place of wonder and excitement within our community.

Experiences

Yash Pahariya

Yash is a 2nd year studying Computer Science. He is involved in Bearcat Motorsports, UC Rocketry, Robotics, and Mountaineering. He likes to learn about automotive engineering, play video games, and enjoys outdoor activities.

Katy Hildebrant

Katy is a 4th year studying Electrical Engineering. She loves watching movies, reading, and doing stuff outside. She has been a member of IEEE at UC since her freshman year and you can usually find her in the IEEE office!

From hackathons to conferences, real-life engineering experiences have the potential to build connections and shape our journies. This section features four articles on FSAE, MakeUC, Cyber@UC CTF, and the International Electron Device Meeting.

Matthew Price

Matthew is a 4th year studying Cybersecurity and Networking in the School of Information Technology. He is also pursuing a Master's in Information Technology here at UC. Matthew is the current President of Cyber@UC and Director of BearcatCTF.

Nicholas Haehn

Nicholas is a 4th year undergraduate student and 1st year graduate student studying Electrical Engineering. He is a member of the University of Cincinnati MIND Lab. Additionally, Nicholas is a member of the Bearcat Bands and the Bearcat Solar Car Team.

My Experience at FSAE Electric 2023

Yash Pahariya, *B.S., Comp. Sci.,*

***Abstract*—Formula SAE Electric (FSAE Electric) was created in 2013 with the purpose of creating a friendly, yet competitive, playground for undergraduate and graduate students to push themselves to design, fabricate, and develop a small formula-style vehicle that performs the best in various metrics such as endurance, efficiency, and outright speed. In this article, I'm sharing the experience I had when attending FSAE Electric 2023 competition and what this experience taught me.**

I. Introduction

FSAE Electric is an electric racing car competition held annually by SAE in the summer. Each year, around 50 college teams compete to design and build the best car that fulfills pre-determined objective performance metrics such as acceleration, handling, and endurance. Fresh after my first year of college, FSAE Electric 2023 at Michigan was such a great experience for me. It has inspired me to share my thoughts, what I learned, and how I have grown from the experience.

II. Background

Bearcat Motorsports (BCMS) is a club at the University of Cincinnati that develops race cars to compete in SAE International's competitions, such as Formula SAE, Baja SAE, and, more recently, FSAE Electric.

In Baja SAE, an off-road vehicle is developed by college teams to withstand the harsh environment of rough terrain. Off-road vehicles are capable of performing in challenging environments and terrains: muddy road, forest trails, etc. They are designed with superior traction, durability and versatility[1]. In Formula SAE, a traditional ICE race car is developed to compete in track environments. ICE vehicles are automobiles powered by conventional, crude oil-derived fuels like gasoline or diesel[2].

FSAE Electric is similar to Formula SAE in that a car is developed to compete in track conditions. However, instead of being powered by an engine, the car is powered by a battery and an electric motor, bringing along with it a unique set of challenges, especially in a performance-oriented environment.

At these competitions, there are three types of events to showcase different aspects of each car. Static events include presentations and inspections to get an overview of the capability and design of a car. Dynamic events include acceleration, endurance, and handling test to determine the performance of a car. An EV Active inspection is used to test the safety systems of the vehicle.

III. Experience

A. First few days

The first few days of FSAE Electric 2023 were primarily for presentations and technical inspections. Judges were presented with design decisions and cost effectiveness regarding the team's car. Technical inspections were exercised over accumulator, mechanical, and other aspects to make sure everything is within safety standard.

Our team arrived at the Michigan International Speedway on Tuesday for presentations and technical inspections. I arrived 2 days later after being advised that the first few days did not necessarily require my attendance. It was the middle of the day after a 3-hour drive from Cleveland, and suddenly, I was overwhelmed as a first-time-goer. A lot was going on, much more than I could comprehend. There were around 70 teams, hailing from US to Canada, Brazil, and Singapore, from prestigious to regular universities: MIT, UC Berkeley, Waterloo, and UPenn, to name a few[3]. Everyone was busy making changes to their cars to pass technical inspections and to be qualified for dynamic events on the weekend.

To my surprise, the sponsors' special cars were displayed around the Speedway as well. Ford had their Mustang Mach-E, Ford GT, and GT MKII. Tesla had their Model 3s and Ys. Rivian had their new R1T. Honda had their Acura NSX Type S with a half of another NSX as its trailer (Figure 1). Multimatic, a Canadian engineering company, even brought a full purpose-built race car - the RT24-P!

Fig. 1: Left to right, top to bottom: Acura NSX Type S, Ford GT MKII, Ford Mustang modified by Roush Performance, Front and Back view of Mazda RT24-P

B. Struggle, Breakthrough, and Accept

Being fresh out of my first year in college, I have to admit that technical skills were not my strong suit. I felt it was best if I stayed in the position of an observer and would try everything I could to help my team.

Unfortunately for us, an issue regarding the roll hoop of the car was spotted during mechanical inspection, preventing us from moving any further. The roll hoop was supposed to be able to protect the driver in an event of a roll over, but in our case, safety was not entirely guaranteed. We eventually had to weld an entire new part to the car. This fix, in fact, would take time due to the constraint associated with dimensions of the new part.

Ideas sparked, plans were implemented, progress was eventually made. But it was just not enough for the short time we had left for that day. Eventually, we decided to keep up our progress for the next day and pushed forward to pass the mechanical inspection. Our hard work was paid off when we passed the mechanical inspection the next day. However, the tears of joy did not exist for so long. EV Active inspection hit us hard right after. We could not pass the safety and electronic system functionality test because of a defective part within the battery management system itself.

When it comes to electric racing cars or electric vehicles (EV) in general, batteries are always a major concern. Overcharging or improperly charging a lithium-cell can cause lithium metal spikes to form on the carbon-graphite anode, which can short out the cell. Then depending on the cell chemistry, fires can be ignited[4]. EV fires are much harder to deal with than gasoline fires[4]. "Tesla has a set guideline for first responders that recommends flooding an EV fire with at least 3000 gallons of water and letting the vehicle burn itself out if water is not effective"[4]. This is to say that, obviously, safety standards must be very high, and building an electric race car fitting every of those requirements is not easy. In the end, although we did not get a chance in our first competition, knowledge and experience were such valuable gifts we gained.

With that understanding, we knew a day was not enough for us to solve our problem. The time crunch on our last day to past all inspections did not give us any time to pull everything together. Defeat was inevitable. Soon enough, we would have to accept it. However, this did not mean the event was all lost. We still had a great time talking to other teams about their cars and understanding their design, watching other teams run their cars in the dynamic events on Friday and Saturday, and talking to sponsors. Shortly after, we learned that we were not the only team that did not pass all inspections. Only around a third of all teams that attended got to compete in dynamic events; the other shared the same boat as us.

C. The End

Friday and Saturday were still ongoing for what was left of the event. We spent the rest of the time watching events such as autocross and endurance, discussing our design, and seeking job opportunities from sponsors. At the end of the event, we cherished the memories we had made over the past few days. Working passionately toward a common goal as a team ultimately creates joy and friendship like never seen before. The last few nights spent together in the same house were the moments that I valued the most. While learning about engineering and building race cars are the main objectives of BCMS, developing relationships and having fun are much more important.

Fig. 2: BCMS's car and driver

Fig. 3: Left to right: 2023 team at Michigan and 2024 team

IV. Thoughts

Overall, I would say this is one of the best club experiences I have had to date! Immersing yourself in a multi-day competition with a team is truly the best time to learn from other people, from yourself, and from the fun. I plan on returning with our team in 2024 with the ultimate goal to pass all inspections and race the car.

V. Conclusion

I highly recommend that anyone interested in the automotive/motorsports industry, as well as anyone interested in any field of engineering, join Bearcat Motorsports whether it be on the ICE, Electric, or Baja team. You can join regardless of your college major and prior knowledge and you will get the opportunity to learn so much, build great connections, and have amazing experiences.

References

[1] Sheer Analytics and Insights. *Off-Road Vehicles Market — Industry Growth, Analysis, Key players, Trend & Forecast*. Accessed: 2024-28-2. URL: https://www.linkedin.com/pulse/off-road-vehicles-market-industry-growth/.

[2] Celine Jerly. *What is an ICE Vehicle? How does it compare to EVs?* Accessed: 2024-28-2. URL: https://www.way.com/blog/what-is-an-ice-vehicle-compare-ev/.

[3] *FSAE Electric 2023*. Accessed: 2024-28-2. URL: https://www.sae.org/binaries/content/assets/cm/content/attend/student-events/results/formula-sae/fsae_ev_2023_results.pdf.

[4] Kevin Clemens. *Racing Safety: Electric Vehicles vs. Gas-Powered Cars*. Accessed: 2024-28-2. URL: https://www.batterytechonline.com/testing-safety/racing-safety-electric-vehicles-vs-gas-powered-cars.

MakeUC Hackathon 2023 Recap

Katy Hildebrant, *B.S., Elec. Eng.*

***Abstract*—MakeUC is the hackathon hosted by the University of Cincinnati's student section of IEEE every fall. The event spans two days with 24 total hours of hacking in which individuals or teams create projects focused on software and/or hardware. Students gain new skills from workshops, learn how to use new tools, and create lasting connections with sponsors and fellow hackers. This year's event took place online and in person at a new venue - the 1819 Innovation Hub.**

I. Background

MakeUC is the hackathon hosted by IEEE at the University of Cincinnati every fall. It is a 24-hour event in which individuals or teams create projects centered around software and/or hardware, learn new skills, and network with each other and event sponsors. The 2023 competition took place on November 4th and 5th online and in-person at the University of Cincinnati's 1819 Innovation Hub. The main activity was "hacking", but a sponsor expo, multiple workshops, MLH-sponsored events, and a Capture The Flag (CTF) cybersecurity challenge hosted by Cyber@UC were among the other fun activities available at the event. In-person participants also had access to a variety of unique benefits at the 1819 Makerspace, which boasts a woodworking shop, soldering stations, 3D printers, laser cutters, and many other tools for professional prototyping and fabrication.

II. Projects

All hackers were encouraged to submit a project in teams of one to four people. A total of 60 projects were submitted by the deadline on November 5th. The event had track prizes (optional focus areas or themes for projects), sponsored prizes, and prizes for first, second, and third overall. Below are the top three projects overall.

1) *Buycott* is an application that shines a light on company's views and standings on current social issues and helps users decide if they want to boycott companies based on this information. The technologies used by the team include Flutter and Flask. The members of this team were Arya Garg, Aniruddhan Ramesh, Ary Sharma, and Kaaustaaub Shankar.
2) *GreenScape* is a specially-trained AI model, meant for use in robots, which can identify biodegradable waste in order to separate it from larger groups of trash. The technologies used by the team include Python and OpenCV. The members of this team were Sudarshan Krishnan, Amit Jadhav, and Abhisar Anand.
3) *Motiv.tech* is a sports community platform aimed at improving team communication and bonding. Technologies used by the team include Python, Swift, and MongoDB. The members of this team were Hamza Khairy, Kaleb Bishop, Jaden Walton, and Jacob Wernke.

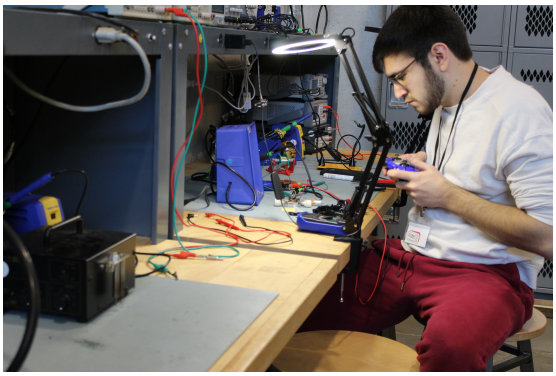


Fig. 1: Hacker at the 1819 Makerspace electronics station

Fig. 2: Hackers collaborating in the 1819 Makerspace

III. Workshops

Workshops are short, hands-on sessions hosted by sponsors where hackers can expand their skills and learn more about topics relating to today's society or to the hackathon themes. This year, workshops were given by the Pittsburg Regional Health Initiative (PRHI), IBM and Elevance Health, and Tata Consultancy Services (TCS).

1) *Patient Safety 101* by PRHI
   In this workshop, a representative from PRHI discussed the importance of increasing public awareness of patient safety, how to engage the next generation of problem-solvers, and how to inspire both academic and industrial organizations to develop technology focused on improving patient safety technology. They also educated hackers on post-pandemic crises such as workforce shortages, healthcare system distrust, and upticks in medical errors. Finally, they gave examples of existing technologies that address patient safety, such as the use of VR in medical training and the use of AI in diagnostics.
2) *AI in Healthcare/Customer Care* by IBM and Elevance Health

In this combined workshop, both IBM and Elevance Health gave insight into the exploding field of AI and its applications in healthcare and customer care. IBM representatives notably described their Watsonx platform which enables developers to train, validate, tune, and deploy generative AI with foundation models and machine learning capabilities. Access to the platform was provided to all hackers, free of charge, for their projects! Additionally, representatives from Elevance Health discussed the evolution of AI, problems in healthcare that have yet to be solved, and how to use technology to overcome real-world issues.

3) *Aerospace Data Tracking* by TCS
In this workshop, a representative from TCS discussed the company's involvement in aerospace data tracking and in dozens of other industries such as banking, education, insurance, energy, and life science. Hackers also had the opportunity to have a group conversation with the representative and ask individualized questions about career paths, life goals, and more.

Fig. 3: Cyber@UC hosting the CTF Challenge

Fig. 4: The MakeUC Organizer Team (not all pictured)

IV. Learn More

If you are interested in learning more about MakeUC, please visit our website at https://makeuc.io. This website is updated for each year's hackathon, so if you are interested in participating in MakeUC 2025, be sure to take a look for important information and links to our other platforms. If you would like to get involved with the event as a sponsor or organizer, or if you have any questions for us, please send an email to info@makeuc.io. If you would like to learn more about the IEEE@UC student branch, please visit our website at https://ieee.uc.edu.

V. Acknowledgements

IEEE at UC is very grateful to the following groups for supporting this event:

- The MakeUC sponsors, for dedicating time, money, and other resources to supporting remarkable opportunities for students.
  - Adonis tier: IBM, Elevance Health
  - Birdwing tier: PRHI, TCS
  - Morpho tier: Infinera, Overleaf, Northrop Grumman, HII, Kinetic Vision, BTS, CincyTech, Fifth Third Bank, Great American Insurance Group, FIS, Tembo, Microsoft, Kao
  - Monarch tier: Wolfram, StickerMule, Standout Stickers, Axure, Taskade, .xyz, Balsamiq, Google Cloud
- The MakeUC judges and mentors, for providing valuable expertise and feedback to the participants.
- The MakeUC hackers, for sharing their creativity with the rest of the community.
- The MakeUC logistics team, for coordinating all organizer teams and hackathon operations.
  - Elaine Mansour (lead), Nathan Braig, Forrest Bushstone, Anton Hoelmer, Mohamed Sameer Imam, Quan Le, Justin Lin, Sushant Padhye, Shashank Sathiyanarayanan, Anay Sehgal
- The MakeUC marketing team, for maintaining our branding, promoting the event, and connecting with hackers.
  - Ziad Oweis (lead), Alex Budnik, Kristin Hildebrant, Anas Khairy, Maanya Naveen, Deeparson Paudel, Mettika Ukey, Shonn Vinchurkar
- The MakeUC sponsorship team, for reaching out and connecting with our sponsors.
  - Greg Muha (lead), Oluwaseun Adekoya, Mohammed Kaif Ali, Natalia Lui, Arleen Monteiro, Khaled Oweis, Shreya Pandey, Dhyan Patel
- The MakeUC technology team, for developing and maintaining our website and Discord server.
  - John Whiting (lead), Jaran Chao, Quan Le, Olivia Leblanc, Cat Luong, Daksh Prajapati, Saarthak Sinha

BearcatCTF 2024 Recap

Matthew Price, *B.S., Cybersec.*

***Abstract*—BearcatCTF 2024 was a massive success for its first year with over 400 students competing globally. As a lead organizer for Cyber@UC's inaugural BearcatCTF 2024 event, I oversaw the creation of promotional material and marketed the event to students at the University of Cincinnati, students in the tri-state area, and students on a worldwide scale. This paper will serve as in-depth discussion of the plans, challenges, and thoughts from BearcatCTF 2024.**

I. Introduction

A CTF, or "capture-the-flag," is a competition that gives participants the opportunity to learn, practice, and try out new skills related to cybersecurity. Each problem is a unique puzzle that can relate to reverse engineering, networking, OSINT, cryptography, forensics, and other skills in cybersecurity. These disciplines, and more, can be found on the OWASP site [1]. Even though Cyber@UC organized successful CTF competitions over the past few years as part of other large-scale hackathons such as MakeUC and RevUC, we decided to run our own independent CTF for the 2023-2024 academic year. Thus, BearcatCTF was made.

II. Planning

The idea for running a 24-hour CTF event originated during a Cyber@UC meeting in the summer of 2023. The idea got lost in planning for the upcoming fall, so we were unable to circle back to the event until September. Planning and developing a timeline did not start until October. We had only five months before our target event date. The following weeks were buzzing with activity because the BearcatCTF planning team was planning and running Cyber@UC's general club activities. Fortunately, our amazing team managed to bring aboard six sponsors, create twenty problems for the CTF, prepare three meals for our competitors, and registered 95 students to compete in-person. Thanks to their continued efforts, we had amassed a total of 400+ competitors worldwide.

BearcatCTF 2024 was held in-person at the 1819 Innovation Hub on February 3rd, 2024 and lasted until February 4th, 2024. Participants in the tri-state area were able to stay overnight at the location to compete in teams and complete CTF challenges to earn as many points as they could. Those who competed in-person were eligible to win over $1,700 worth of prizes. What makes BearcatCTF special is the ability for people around the world to compete as well, making the event accessible to anyone with a computer.

III. BearcatCTF 2024

We kicked off the event with an opening ceremony led by me, Matthew Price. During the kickoff, I introduced the sponsors of BearcatCTF and our keynote speaker, Dr. Marc Cahay, spoke about the importance of education and careers in Cybersecurity (Fig. 1)[2].

Fig. 1: Matthew Price announcing the start of BearcatCTF 2024.

This was my first time speaking in front of a lot of people in a professional setting, so I was a bit nervous. Fortunately, with the help of our volunteers, I was able to prepare myself properly and kicked off the event.

Following the opening ceremony, participants had options on what they could do next. For the next two hours, in-person participants were able to talk with representatives from our sponsors at the sponsor expo. Here, they could learn about the work each company and individual does, network with professionals, and pick up some free swag. Cyber@UC's CTF Team Lead, Chad Lape, held a CTF 101 session for participants that were new to CTF competitions. Chad walked through a sample CTF problem and showed off various tools that the participants would need to compete in BearcatCTF.

The 24-hour countdown started, and competitors broke out into their teams to work on the problems collaboratively. According to numerous comments from competitors: "This teamwork aspect of the CTF was by far their most enjoyable experience throughout the event" (Fig. 2-3)[2].

Fig. 2: Four participants collaborating at BearcatCTF 2024.

Fig. 3: Two participants collaborating at BearcatCTF 2024.

During the 24 hours, competitors asked questions, solved problems, ate meals, and, overall, had a great time. I closed out the event proclaiming the winners and congratulating everyone for their hard work through the night and into the morning.

IV. Thoughts

After five months of planning, coordinating, and marketing up until BearcatCTF 2024, it was a weird feeling when it concluded. The entire planning team and I believed that the event was a roaring success-and so did the competitors. This CTF event would not have been possible without the countless hours our team put into making promotional material, creating problems, coordinating with sponsors, and setting up the event's infrastructure. I want to give a huge thanks to Cyber@UC alumnis Cole Duffy, Chad Lape, and Josh Hale who put so much work into BearcatCTF in their last year at UC. Running an event to this scale with only five months of planning should not have been possible, but the collective effort of our entire Cyber@UC board and members allowed us to make this happen.

V. Conclusion

Even though BearcatCTF 2024 was considered a success, there is always room for improvement. Several competitors expressed their aspiration to compete next year, and we are attempting to make this wish come true. I will be assuming the role of Director for BearcatCTF 2025 and planning has already started. We are really looking forward to making the event bigger and better in the upcoming year.

References

[1] *OWASP*. Accessed: 2024-04-17. URL: https://owasp.org/.

[2] *Photographs taken by Lyriq Davis*. Accessed: 2024-04-17.

2023 International Electron Device Meeting

Nicholas Haehn , *B.S., Elec. Eng., M.S., Elec. Eng.*

***Abstract*—Following the end of the Fall 2023 semester, I had the privilege to travel to the 2023 International Electron Device Meeting (IEDM) with Siddharth Barve, Connor Socolik, Greg Muha, and our research advisor, Dr. Rashmi Jha, at San Francisco, California. I was given this opportunity because I co-authored a paper focused on neuromorphic architectures with Siddharth Barve, Nicholas Haehn, Connor Socolik, Aaron Ruen, Joshua Mayersky, Amber Reed, Kevin Leedy, and Rashmi Jha that had the pleasure of being presented at the conference. During the IEDM, I attended multiple groundbreaking sessions across a wide variety of topics. This paper will discuss my experience attending the IEDM Conference, the presentation of our paper at the conference, and other experiences on the trip.**

I. Introduction

IEDM is a conference held in San Francisco, California by the Electron Device Society (EDS) and the leading conference in reporting breakthroughs in semiconductor and electronic device technology, design, manufacturing, physics, and modeling [1]. "Moore's Law", the observation that the number of transistors in an an integrated circuit doubles every two years [2], was introduced at IEDM in 1965 by Gordon Moore following its initial publication in *Electronics Magazine*.

IEDM contained 41 sessions with focuses in DRAM devices, 2D Materials, Ferroelectric technology advancements, 3D stacking technologies, RRAM arrays, Compute-in-Memory for Deep Learning, Artificial Intelligence (AI), Quantum technology, High Frequency/RF Devices, image sensors, and more. Many of these sessions were held by top industry presenters and universities worldwide including TSMC, Intel, ASML, IMEC, IBM, Micron, and many more and attended by over 1,900 people.

II. IEDM Conference Presentations

At the University of Cincinnati MIND Lab, my research is currently focused on the characterization and applications of ferroelectric devices. Ferroelectric devices are materials that can have a polarization effect applied through the application of an electric field to the device [3]. These devices can be used as a singular ferroelectric capacitor, on the gate of a transistor known as a Ferroelectric Field Effect Transistor (FeFET), on the source or drain of a transistor (FeRAM), or in other configurations. Commonly, ferroelectrics are explored as memory devices or for applications in AI [4].

Across the entire IEDM conference, I attended over 30 presentations in a wide variety of topics. Since my research is focused on ferroelectric devices, I attended many sessions involving ferroelectrics. From these presentations, I learned more about ferroelectric wakeup, fatigue, and recovery characteristics and measurement schemes [3, 5]. Understanding these concepts will greatly improve my measurement techniques in my own research while providing baseline measurements for my future work. Additionally, I learned about new ferroelectric materials and combinations currently used to achieve large memory windows and endurance [6]. I also learned about fabrication and integration techniques that allow for 3D stackable FeFETs, and, specifically, a breakthrough in NVDRAM by Micron when creating a 32 Gb stacked non-volatile ferroelectric memory with performance for AI workloads [4]. Furthermore, I learned about the non-destructive read-out of ferroelectric capacitors [7]. These devices traditionally had destructive read-out, but this presentation described the expansion of the application space of ferroelectric capacitors. This presentation is shown in Figure 1.

Fig. 1: A conference presentation on Non-Destructive Read Endurance of Ferroelectric Capacitors.

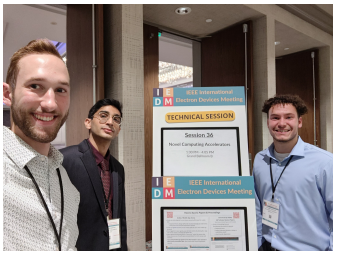
Other notable talks at the conference discussed the future of generative AI hardware both in terms of the uses within in the design industry and the improvement of the hardware itself [8]. Fabrication techniques using EUV were discussed which showed how the industry can be more energy efficient [9]. Device improvements were also shown in DRAM, showing higher scalability, retention, and faster operation at smaller sizes [10].

III. Co-Authoring a Paper for IEDM

The opportunity to attend IEDM was the acceptance of our paper entitled "Power-Efficient Clustering Using Programmable V_T FETs in Neuromorphic Architectures" by Siddharth Barve, Nicholas Haehn, Connor Socolik, Aaron Ruen, Joshua Mayersky, Amber Reed, Kevin Leedy, and Rashmi Jha [11]. Siddharth Barve worked on the architecture portion and I focused on testing and characterizing the ferroelectric capacitors. Connor Socolik worked on testing the FeFETs for the paper. Aaron Ruen assisted in testing and Joshua Mayersky, Amber Reed, and Kevin Leedy worked to fabricate the devices in collaboration with the Air Force Research Laboratory. Rashmi Jha advised the project.

In the paper, we presented a nueromorphic architecture that allows for Squared Euclidean Distance calculation using

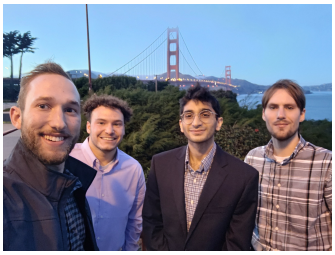
(a)



(b)

Fig. 2: (a) Siddharth Barve presenting "Power-Efficient Clustering Using Programmable V_T FETs in Neuromorphic Architectures". (b) Nicholas Haehn, Siddharth Barve, and Connor Socolik next to the sign for our paper session.

(a)



(b)

Fig. 3: (a) Waymo Self-Driving Car taken to the (b) Golden Gate bridge with other University of Cincinnati Students.

programmable V_T FETs. In the experimental validation of the architecture, we focused on the use of Barium Titanate-based ferroelectrics for the programmable V_T FETs. Through this work, I learned about the characterization techniques of these devices. From the data, we were able to show a relation between the long endurance and strong retention of the ferroelectric capacitors as well as proper functioning as FeFETs. The architecture simulated properly, correctly showing calculations of the Squared Euclidean Distance which is extremely useful for Deep Neural Network (DNN) applications.

Siddharth Barve presented the paper in the session entitled "Novel Computing Accelerators" at the IEDM conference. We were able to interact with other universities and industry to gain feedback on the architecture, device performance, and testing. Figure 2a shows Siddharth Barve presenting the work.

IV. Other Experiences

While attending the conference, we participated in a number of different experiences. I connected with people from companies across the industry from researchers and suppliers to device manufacturers. These connections will be vital in my future job search. These interactions were also helpful in gaining more insights and perspectives on our work and other routes we could follow.

Additionally, we were able to experience parts of the city during the conference breaks. Of note, we tested the Waymo Autonomous Vehicles (Figure 3a) in San Francisco to go to the Golden Gate Bridge (Figure 3b). These vehicles operate without a driver through the use of LIDAR and other tracking systems. This experience allowed us to experience some of the applications currently in testing and in use for the devices and algorithms being researched.

V. Conclusions

Overall, my experience at the IEDM conference allowed me to gain more perspectives on my research work as I interacted with other engineers, using the knowledge learned from their research to further improve my own. I connected with major companies and universities at the conference while experiencing the process of preparing a paper for a high-level conference alongside my co-authors.

References

[1] *IEDM Overview*. 2023. URL: https://www.ieee-iedm.org/about-overview.

[2] Gordon E Moore. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." In: vol. 11. 3. 2006, pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860.

[3] C. Cho et al. "Wake-Up of Ultrathin Ferroelectric Hf0.5Zr0.5O2: The Origin and Physical Modeling". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[4] N. Ramaswamy et al. "NVDRAM: A 32Gb Dual Layer 3D Stacked Non-volatile Ferroelectric Memory with Near-DRAM Performance for Demanding AI Workloads". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[5] J. Chen et al. "Physical Origin of Recovable Ferroelectric Fatigue and Recovery for Doped-HfO2: Toward Endurance Immunity". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[6] M. Zeng et al. "First Demonstration of Annealing-Free Top Gate La:HZO-IGZO FeFET with Record Memory Window and Endurance". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[7] S. Mukherjee et al. "Pulse-Based Capacitive Memory Window with High Non-Destructive Read Endurance in Fully BEOL Compatible Ferroelectric Capacitors". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[8] S. Naffziger. "Innovations For Energy Efficient Generative AI". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[9] T. Thijssen et al. "EUV Energy Efficiency". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[10] S. Ryu et al. "A Workfunction Engineered Middle-Silicon-TiN Gate (MSTG) Cell Transistor in 16Gbit DRAM for High Scalability and Long Data Retention". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

[11] Siddharth Barve et al. "Power-Efficient Clustering Using Programmable VT FETs in Neuromorphic Architectures". In: *2023 International Electron Devices Meeting (IEDM)*. 2023, Accepted.

Projects

To innovate, from idea to project, creates immense opportunity. This section features two articles about non-invasive bio-sensors that provide real-time health monitoring and a Chrome extension that enables anonymous communication between LLMs and webpages.

Grant Jolly

Grant is a 5th year majoring in Biomedical Engineering. He contributes as a student researcher at the Novel Device Lab (NDL). Outside the lab, Grant likes to run, play golf and basketball, and manage a sneaker business.

Arnav Komaragiri

Arnav is a 2024 Computer Science graduate. He is interested in Large Language Models (LLMs), researching automated red-teaming of LLMs, Retrieval-Augmented-Generation (RAG) systems, and LLMs as compression.

Transformative Wearable Devices: Non-Invasive Biosensors for Real-Time Health Monitoring

Grant Jolly, B.S., Biomed. Eng.

***Abstract*—We are leading the way in developing Electrochemical Aptamer-Based (E-AB) sensors at the University of Cincinnati's Novel Device Lab (NDL), with hopes to revolutionize the field of continuous molecular monitoring. This study summarizes the development of E-AB sensors from concept to almost commercialization, emphasizing their usefulness, design, and creative problem-solving capabilities. With careful surface chemistry and component optimization, our graduate and student researcher team aims to improve the stability, specificity, and lifetime of sensors. These developments pave the way for using E-AB sensors in non-invasive, real-time health monitoring. By sharing our knowledge and advancements in biosensing, this paper hopes to expand the application of continuous molecular monitoring instruments.**

I. Introduction

The Novel Device Lab predicts a change in the medical diagnostics environment where traditional, costly, intrusive blood tests are no longer used for illness management, disease prevention, or monitoring critical molecules. "Recent Progress in Intelligent Wearable Sensors for Health Monitoring and Wound Healing Based on Biofluids"[1]. Our goal is to develop a wearable biosensor that can continuously, non-invasively, and economically, monitor a variety of analytes, such as medications, biomolecules, and more. We aim to replace sporadic blood tests with accurate, real-time health data streams by integrating Electrochemical Aptamer-Based (E-AB) sensor technology. This study describes our lab's efforts to overcome the current limits in biosensor technology to expand the use of continuous molecular monitoring devices and establish new benchmarks for patient care and the management of chronic diseases.

- **Background definitions** The following terms are crucial to comprehend our work with biosensors and the broader ramifications of our study. These simple definitions will ensure that readers who are unfamiliar with the subject matter can understand it:

1) Continuous Molecular Monitoring: The method of utilizing wearable or implantable devices to continuously measure the concentrations of various chemicals present in the body. It can give current information regarding a person's health and how they respond to treatments.
2) Biosensor: A device that detects certain compounds in the body and converts their presence into a quantifiable signal using biological components such as enzymes, antibodies, and aptamers.
3) Analyte: A material whose chemical components are being tested and identified as analytes. In this paper, it refers to the essential biomarkers such as medications and bio-molecules that the biosensor is intended to identify and quantify within the body.
4) Interstitial Fluid (ISF): The fluid that envelops and fills the gaps between bodily cells is called an ISF. It usually comes from blood plasma and mainly consists of blood, waste products, water, electrolytes, nutrients, and other elements. "Continuous molecular monitoring of human dermal interstitial fluid with microneedle-enabled electrochemical aptamer sensors"[2]. Additionally, it is comprised of extracellular matrix molecules and interstitial proteins which are not present in blood.
5) Aptamer: Single or double-sided nucleotide molecules capable of selectively binding to particular targets through intramolecular forces enabling high specificity, reversibility, and precision in media such as blood, serum, or interstitial fluid.
6) Electrochemical process: A process that involves both electrical and chemical reactions used by our sensors to convert binding events in the body into a signal we can interpret.
7) Electrochemical Aptamer-Based Sensor A biosensor that translates particular chemical interactions into quantifiable electrical signals transduced by these interactions in different biological mediums by employing aptamer technology.
8) Self Assembled Monolayer (SAM) A single molecular layer that spontaneously develops on a substrate and is crucial for modifying surface characteristics in sensor applications. It is typified by the organized assembly of molecules, such as alkanethiols, on gold surfaces.
9) Blocking Layer: A selectively permeable layer applied to sensor surfaces to prevent non-specific binding and interference from non-target molecules, thereby enhancing the sensor's specificity and sensitivity to its intended analyte.

II. Overview

Wearable glucose monitors are the gold standard for continuous molecular monitoring devices and serve as significant evidence of the high level of technological advancements used today. These devices test glucose levels painlessly by inserting a microneedle into the subcutaneous interstitial fluid (ISF). Now, they are so inexpensive that they are disposable and factory-calibrated to last for weeks "Wearable biosensors for healthcare monitoring"[3]. Unfortunately, their use is limited to high-concentration target analytes such as glucose. This raises the question: what about monitoring for other long-term conditions and different health requirements? This is the

motivation behind our work at the Novel Device Lab. By technologically transforming Electrochemical Aptamer-Based (E-AB) sensors into biosensors, we can expand the possibilities of molecular monitoring beyond the industry standard glucose monitors.

A. Clinical Significance

Our study addresses the requirement for ongoing, non-invasive analyte monitoring in ISF. The similarity of analyte concentration between the ISF and blood, which reflects changes in blood chemistry without requiring invasive blood sampling, gives clinical significance to the measurement"Continuous molecular monitoring of human dermal interstitial fluid with microneedle-enabled electrochemical aptamer sensors"[2]. This creates an opportunity to simplify and automate the monitoring of analytes by non-invasive, or minimally invasive, techniques such as implantable devices that measure ISF in subcutaneous tissue or wearable devices that measure ISF through the skin. The implantable E-AB sensors intended for long-term usage in this medium are the subject of this project.

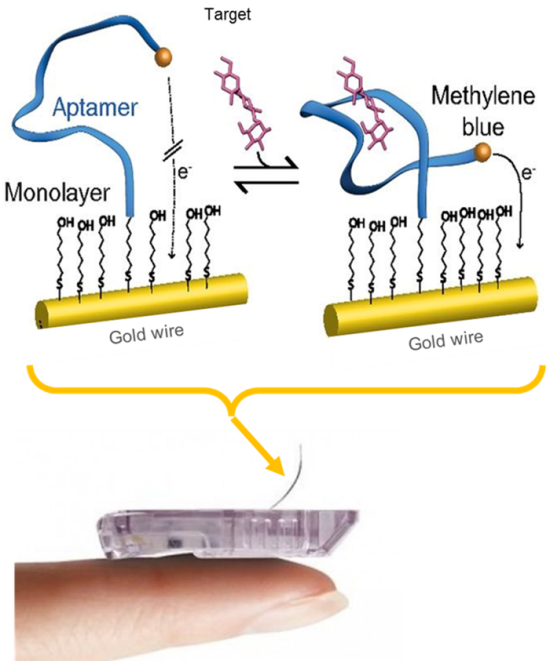


Fig. 1: Novel Biosensor Design utilizing E-AB Technology.

B. Design

The design of the E-AB sensors is shown in Figure 1. A conductive layer, usually gold, is deposited onto a substrate which chemically bonds to the aptamer before adding an additional Self-Assembled Monolayer (SAM) like 6-mercapto-1-hexanol (MCH). The most essential component of the sensor's operation is the redox reporter molecule. It is used to easily convert binding events into electrical signals that can be measured.

III. What are E-AB Sensors, and How Do They Work?

Electrochemical Aptamer-Based sensors represent a potential leap forward in the field of bio-sensing, a technology that the Novel Device Lab is actively working to develop. These sensors are made to precisely and selectively attach to target analytes in various biofluids "Detection and beyond: challenges and advances in aptamer-based biosensors"[4].

The core of an E-AB sensor's functionality lies in its ability to translate molecular interactions into electrical signals. When an aptamer on the sensor binds to its target analyte, it undergoes a structural transformation. This change in shape affects the flow of electrons between a redox reporter molecule, attached to one end of the aptamer, and the sensor's electrode surface. Because of this conformational shift, the electron transfer becomes more noticeable as the reporter approaches the electrode, producing a detectable current. This current directly correlates with the concentration of the target analyte, providing a quantitative readout of its levels in the biofluid.

Analyte trends can be tracked by changes in a measured current by altering the voltage range to the redox reporter and monitoring the resulting current. This crucial stage in the functionality of E-AB sensors is managed by an instrument called a potentiostat. This apparatus sends a particular voltage to the sensor's electrode, initiating a movement of electrons to identify the target biofluid's analyte concentration.

The versatility of E-AB sensors is further demonstrated in their infinite customizability that can detect an infinite number of analytes, achieved by changing their nucleotide sequence. Unlike traditional biosensors, which are generally limited to a one-time catalytic interaction with the analyte, the binding between an aptamer and its target in E-AB sensors is reversible, allowing for repeated use without analyte consumption.

IV. The Current Landscape of E-AB Technology

A. Challenges and Strategies for Enhancing Sensor Longevity and Stability

Although E-AB sensors hold great potential for continuous molecular monitoring, issues with sensor lifespan, especially when exposed to complex biofluids at body temperature, have prevented their broad clinical application"From the Beaker to the Body: Translational Challenges for Electrochemical, Aptamer-Based Sensors"[5]. The Novel Device Lab has addressed these issues by developing strategies to enhance sensor performance in demanding biological environments.

B. Understanding and Mitigating Monolayer Degradation

The deterioration of self-assembled monolayers, which are crucial for sensor specificity and signal-to-noise ratio, is a significant problem for the stability of E-AB sensors. Alkylthiolate oxidation and disulfide production are two processes of degradation. Repeated scanning and the body's electrical and thermal strains can sometimes worsen this process. Our lab aims to gain a greater knowledge of these mechanisms to develop better strategies that increase the operational life of sensors "Week-Long Operation of Electrochemical Aptamer

Sensors: New Insights into Self-Assembled Monolayer Degradation Mechanisms and Solutions for Stability in Serum at Body Temperature"[6].

C. Addressing Biofouling through Protective Strategies

The unwelcome build-up of proteins and other biomolecules on sensor surfaces, known as biofouling, poses a severe threat to the physical integrity of aptamer sensors. This impacts the aptamer's capacity to attach to target molecules and detect change. In the lab, we study and create antifouling protective surface chemistries to minimize the effect of biofouling on sensor performance.

D. Optimizing Sensor Design for Clinical Applications

The strong and dependable performance of E-AB sensors is essential for clinical application. To balance sensitivity and stability, our team has been honing the sensor's design by focusing on electrode material selection, aptamer optimization, and monolayer integrity to develop a commercially viable E-AB sensor platform.

V. Preliminary Results and Discussion

A. Sensor Longevity and Stability

Our empirical findings demonstrate that the functional stability of E-AB sensors may be extended through software-enabled peak"Voltammetry Peak Tracking for Longer-Lasting and Reference-Electrode-Free Electrochemical Biosensors"[7] tracking and monolayer composition tuning. These results support the hypothesis that sensor lifespan can be significantly increased since specific sensors function correctly, in serum, at body temperature for more than a week. This is a significant improvement over the previous benchmarks of a few hours.

By increasing van der Waals interactions between monolayer components and optimizing electrochemical measurements, we have successfully mitigated these degradation pathways, achieving sensor functionality for over a week in serum at 37°C "Week-Long Operation of Electrochemical Aptamer Sensors: New Insights into Self-Assembled Monolayer Degradation Mechanisms and Solutions for Stability in Serum at Body Temperature"[6].

B. Degradation and Biofouling Mitigation

Our theoretical models have been confirmed by successfully reducing biofouling by applying zwitterionic materials. Zwitterionic compounds modify sensor surfaces to improve biocompatibility and prevent the non-specific adsorption of proteins and other biomolecules. These compounds have both positive and negative charges within the same structure. These findings are significant because they imply that material science and chemical techniques can be used to reduce biofouling. Hydrogel coatings have also been investigated to protect the sensor surface and maintain the sensor sensitivity and specificity in biofluids for prolonged periods "Week-Long Operation of Electrochemical Aptamer Sensors: New Insights into Self-Assembled Monolayer Degradation Mechanisms and Solutions for Stability in Serum at Body Temperature"[6].

VI. Conclusion

Our research at the Novel Device Lab has led to significant advancements in the development of E-AB sensors, demonstrating the potential to revolutionize continuous, non-invasive health monitoring. A significant step toward these sensor's integration into clinical practice and customized healthcare has been demonstrated by their enhanced stability, longevity, and resistance to biofouling.

The path to commercializing E-AB sensors involves a multifaceted approach, encompassing material science, molecular biology, and biomedical engineering. The approaches we have employed, from software-enabled peak tracking to the application of alternative blocking layer materials for biofouling mitigation, have not only validated our initial hypotheses, but also presented new avenues for further research. The versatility of this technology in meeting the objectives of medical diagnostics is demonstrated by the adaptability of E-AB sensors to a wide range of analytes and their improved operating stability.

References

[1] S. Cheng et al. "Recent Progress in Intelligent Wearable Sensors for Health Monitoring and Wound Healing Based on Biofluids". In: *Front. Bioeng. Biotechnol.* 9.765987 (Nov. 2021). DOI: 10.3389/fbioe.2021.765987.

[2] M. Friedel et al. "Continuous molecular monitoring of human dermal interstitial fluid with microneedle-enabled electrochemical aptamer sensors". In: *Lab Chip* 23.14 (July 2023), pp. 3289–3299. DOI: 10.1039/D3LC00210A.

[3] J. Kim et al. "Wearable biosensors for healthcare monitoring". In: *Nat Biotechnol* 37.4 (Apr. 2019). DOI: 10.1038/s41587-019-0045-y.

[4] H. Yoo et al. "Detection and beyond: challenges and advances in aptamer-based biosensors". In: *Mater. Adv.* 1 (Oct. 2020), pp. 2663–2687. DOI: 10.1039/D0MA00639D.

[5] N. Arroyo-Currás et al. "From the Beaker to the Body: Translational Challenges for Electrochemical, Aptamer-Based Sensors". In: *Anal. Methods* 12 (Feb. 2020), pp. 1288–1310. DOI: 10.1039/D0AY00026D.

[6] Z. Watkins et al. "Week-Long Operation of Electrochemical Aptamer Sensors: New Insights into Self-Assembled Monolayer Degradation Mechanisms and Solutions for Stability in Serum at Body Temperature". In: *ACS Sens.* 8.8 (Mar. 2023), pp. 1119–1131. DOI: 10.1021/acssensors.2c02403.

[7] A. McHenry et al. "Voltammetry Peak Tracking for Longer-Lasting and Reference-Electrode-Free Electrochemical Biosensors". In: *Biosensors* 12.782 (Sept. 2022). DOI: 10.3390/bios12100782.

CHAAP: A Hackathon Experience

Arnav Chandra Komaragiri, *B.S., Comp. Sci.,*

***Abstract*—RevolutionUC is a hackathon held in the spring semester where students have 24 hours to develop a technology solution. During this event, my team and I developed CHAAP: CHat with Any Anonymous Page, a Chrome extension that allowed Large Language Models (LLMs) to talk to the given website and perform Question Answering and other AI workloads. CHAAP was the result of a very deliberate system design process that allowed our team to solve a hackathon project in a much more structured software development manner. Additionally, we implemented an anonymization algorithm that allowed users to use large API LLMs without worrying about private data leakage. By the end of the event, CHAAP won the 1st Place Prize for the entire event, the Best Use of AWS, and the Best Use of Data Science and Analytics. This paper will outline our team's hackathon experience and CHAAP's system architecture, shedding light on the work that can come from a hackathon.**

I. Introduction

CHAAP was an app designed to bring the benefits of Large Language Models (LLMs) to everyone while mitigating many of their threats which is a critical issue in today's technological climate. With the massive pre-training datasets needed to create LLM "foundation models", there are massive concerns about where exactly the data to power these LLMs comes from. Recently, the New York Times sued OpenAI for using NYT articles to train ChatGPT, violating copyright restrictions [1]. As such, it was clear that whatever AI system we developed at the hackathon, we needed to make sure it was fair to its users.

It is this desire that led to CHAAP, a Chrome extension that allows users to apply LLMs to webpages to parse through data, but still respect the data privacy of the page content. To achieve this, we chose to implement a Chrome extension where the majority of the files are implemented in a local cache as opposed to pinging a website the user does not own. Additionally, we allow the user to pick the LLM back-end they want between a local OLLAMA [2], an AWS pipeline running LLaMA2-7B [3], or ChatGPT3.5-Turbo [4]. Finally, we also developed an anonymization algorithm called HEMLock (Hidden Enciphered Meaning Lock) by borrowing ideas from differential privacy and masked language modeling (MLM) to scramble text sent to API LLMs like OpenAI. With these features, CHAAP delivered a robust feature set that delivered on its aims, resulting in a highly effective app that was very competitive at the event.

II. Hackathon Experience and Timeline

Our team spent a large portion of this hackathon using a structured approach to building our solution. As such, we took many steps to ensure our time during the event would be organized and well executed. For instance, our brainstorming process started almost a week before the event. We met to discuss our vision for the project and the components we would like to implement during the event. During this time, we designed a rough system architecture on paper that outlined CHAAP's components: webpage/PDF interfacing, LLM inference, prompt libraries, memory (conversation history, database), anonymization, and the chat interface. We also allocated teams for each component: Andrew would work on LLM inference and prompting, I would work with Andrew on prompting and memory while doing anonymization, Bek would work on the front-end chat interface and Chrome extensions, and Jack would work on webpage/PDF interfacing and HTML parsing.

During the event, we were quickly introduced to Murphy's Law: anything that can go wrong will go wrong. Setting up the LLM inference through AWS was highly complicated. We spent a massive amount of time configuring the LLM and ensuring everything worked. Our teammate, Andrew, spent so much time with the AWS Solutions Architects that he appeared for a period of $\sim$4-5s on the venue's timelapse during the closing ceremony, which illustrates the degree of difficulty that singular component took. Additionally, we faced a great deal of difficulty configuring Tembo to do Retrieval Augmented Generation. Our initial timeline had us completing our Minimum Viable Project (MVP) by 10 PM on the first day, but it took us until nearly midnight to just configure the back-end for anonymized Retrieval Augmented Generation. Furthermore, we had several issues with API key management. This occurred because keys were accidentally committed to the GitHub repository at random. Finally, it took another 3 hours, after we got the back-end done, to finish the front-end because we encountered several difficulties with configuring permissions in Chrome extensions.

Despite these issues, our team managed to deliver on the majority of our initial project specifications. Andrew got the LLaMA-2 7B [3] pipeline running on AWS and somehow managed to get pipelines running for both OLLAMA [2] and GPT3.5 [4]. Jack and Bek managed to get the front-end and HTML parsing work done which enabled our system to parse the shown webpage and nested subpages. Unfortunately, we could not get the PDF parsing done, which was a big point we wanted to execute, but the resulting product was more than enough. A large number of judges greatly enjoyed our product and we went on to be very competitive for prizes during the event. It was not what we originally planned, but our engineering process worked very well.

III. CHAAP System Architecture

This section will now outline the overall structure of CHAAP. The goal of this section is not to fully detail the engineering structure of CHAAP. This section is meant to

provide context on many of the discussions in this paper and outline the work the CHAAP team devoted towards their product. This section will specifically outline the LLM inference component, the HEMLock anonymization algorithm, and the Chrome extension/webpage interface.

A. System Outline

CHAAP is made up of three primary components: front-end (webpage interfacing), unstructured data processing (retrieval, memory, anonymization), and LLM inference. These components utilize many different technologies: HuggingFace, Presidio, OLLAMA, OpenAI, AWS, and Chrome Extensions. These technologies are arranged in the system architecture below, shown below in Figure 1.

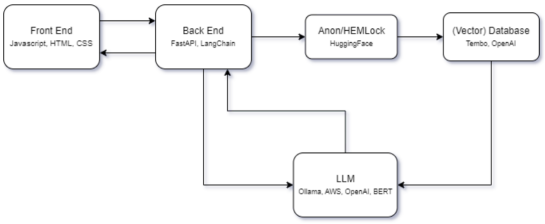


Fig. 1: CHAAP Architecture

B. Question Answering with LLMs and RAG

CHAAP uses a Retrieval Augmented Generation (RAG) paradigm for single-question-answering problems. Specifically, CHAAP uses Tembo to run a Postgres instance that stores chunks of webpages and runs a cron job to generate text embeddings for vector retrieval. This results in a type of "semantic search" where the question asked to the LLM is used to retrieve relevant chunks of the queried webpage as context for the LLM to answer the given question. Contrary to many other apps, CHAAP does not use LangChain to handle retrieval and LLM inference. This is because the LangChain interface locks the app structure into a fairly limited paradigm, however, many of the functionalities LangChain could provide beyond basic text chunking can be trivially implemented with far more flexibility.

With retrieval handled by Tembo, CHAAP uses a prompt library to apply templates on the retrieved content before sending the text into the chosen LLM. The choice of LLM is handled via a drop-down interface, with the appropriate back-end loaded and cached when the user selects a specific back-end. CHAAP offers 3 different back-ends: OLLAMA [2], AWS, or GPT3.5 [4]. The OLLAMA server can be locally configured by the user with CHAAP pinging a specific configurable URL on the extension launch. AWS runs a LLaMA-2 7B parameter model [3], offering a compute-lightweight alternative to OLLAMA inference. Finally, for users wanting a stronger language model, we allowed the use of GPT3.5 given an API key, which, for the hackathon, was hard-coded, but this functionality can be opened to user input.

C. HEMLock: AI-Powered Text Anonymization

With CHAAP, we also released HEMLock: a text anonymization algorithm that can eliminate unique user information and serve as a defense against pre-training data theft. HEMLock works by turning the limitations of local language models into an advantage. This effectively truncates the text distribution to the limited information a local language model can generate. More specifically, HEMLock operates in two stages:

1) Explicit Personally Identifiable Info (PII) Masking
2) Differentially Private Random Masking

The first stage performs an explicit masking stage, identifying parts of the text that contain personally identifiable information. This stage is requisite of a necessary evil, guaranteeing that there is nothing explicit that can tie the piece of text to any people/locations/etc. We use Presidio [5] to segment out personally identifiable information and replace segmented entities with a mask token. We do not use substitution at this stage. Instead, we leave the substitution task to the downstream language model.

The second stage is where HEMLock begins to differentiate itself from other anonymization algorithms. HEMLock samples a Bernoulli distribution for every word/token in the sentence, flagging each word to be masked with some probability p. We sampled indices to mask via:

$$i \sim Bernoulli(p), |i| = N, i \in \{0, 1\}$$

In this case, i is the mask of words/tokens to mask. If $i = 1$, we mask the word/token at that location. This mask is then unioned with the mask from the explicit PII scrubbing and provides a list of locations to reconstruct using a language model.

Once we have compiled a list of all the locations to "anonymize", we replace the words/tokens at those locations with mask tokens. We then use HuggingFace with the DistilBERT [6] masked language model to replace the masked token with whatever the DistilBERT [6] model predicts. We also randomize the order in which tokens are substituted, resulting in $O(N!)$ possible orderings. This large amount of orderings, along with the "plausible deniability" introduced by having DistilBERT [6] perform masked language modeling, results in an anonymization algorithm that can effectively strip out style and personally identifiable information while retaining enough semantic information to answer the question. Finally, the DistilBERT [6] model is quite lightweight, comfortably fitting into a laptop's RAM restrictions and running comfortably on the CPU. Analysis of this algorithm is not yet complete and we leave this work to future papers.

HEMLock is applied to text content before it is ever sent to any database or LLM systems, guaranteeing that private data is never saved. This way, CHAAP always makes sure user data is safe which puts power back in the hands of the user.

D. Chrome Webpage Interface

CHAAP pulls data from the current webpage through our Chrome extension then loads the webpage content as HTML. It then parses out plaintext fields and linked pages. From there,

we pull in the first K links found in the parsing process and load data from those pages which loads the full webpage context into memory. This data is then sent to the memory module to be chunked, anonymized, and loaded into RAG. CHAAP provides a simple chat interface to communicate with the LLM, providing an interface to converse with the LLM and process the page data.

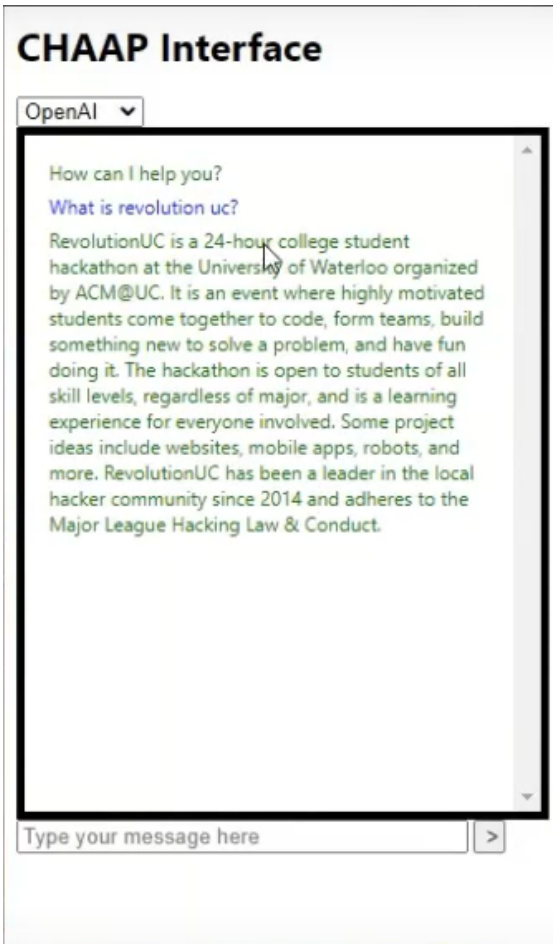


Fig. 2: CHAAP User Interface

IV. Conclusion

This paper outlined my experience building CHAAP at the RevolutionUC hackathon and described the system our team built, our plan going into the event, and the difficulties we faced. I am incredibly proud of our team and what we were able to accomplish and I look forward to working with this system in the future. I would like to extend an enormous thanks to the team: Andrew Gerstenslager, Bekarys Dukenbaev, and Jack McKain. Everyone did amazing and I am truly, incredibly honored to be a part of this team. We hope that CHAAP goes on to deliver the power of AI to people all over the world while mitigating many of the damages AI has inflicted on people and ensuring that the technology we exalt as ”world-changing” changes our world in a positive light. To quote Alan Turing, ”We can only see a short distance ahead, but we can see plenty there that needs to be done” [7].

References

[1] https://www.nytimes.com/by/ryan-mac. *The Times Sues OpenAI and Microsoft Over A.I. Use of Copyrighted Work — nytimes.com*.

[2] *Ollama — ollama.com*. https://ollama.com/.

[3] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307 . 09288 [cs.CL].

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[5] *GitHub - microsoft/presidio: Context aware, pluggable and customizable data protection and de-identification SDK for text and images — github.com*. https://github.com/microsoft/presidio. [Accessed 22-03-2024].

[6] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2020. arXiv: 1910.01108 [cs.CL].

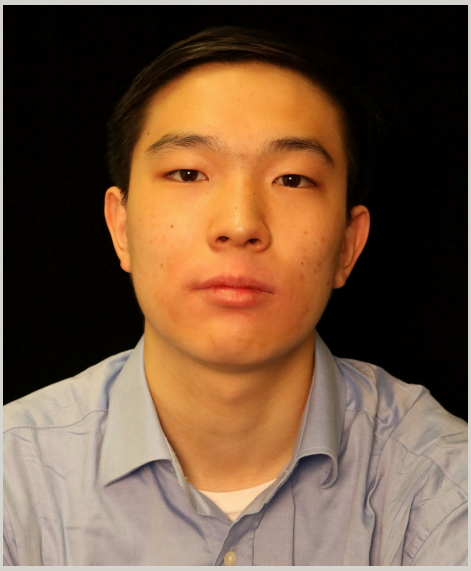
[7] A. M. TURING. “I.—COMPUTING MACHINERY AND INTELLIGENCE”. In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

Research

Research is the root of all advancement. This section features three articles on sensors in surgery robots, generative AI usage in students, and defining the identity of a programming language.

Anna Franchi

Anna is a 4th year studying Electrical Engineering. She is minoring in Robotics & Automation and wrote her article in her class, Haptics & Collaborative Robotics.



Jaran Chao

Jaran is a 3rd year studying Computer Science. He enjoys learning and discussing programming language theory and development, mathematics as it pertains to computing, and image processing.

Sensors In Surgery Robots: A Review

Anna Franchi, *B.S., Elec. Eng.*

***Abstract*—This article describes the nuances of robots in a surgical environment and, specifically, how sensors impact the performance of a robot. In this article, we will discuss how these robots work by understanding how the robot and surgeon interact through haptics, understanding the specific force and torque sensors used in these robots, and understanding how degrees of freedom (DoF) impact these sensors.**

I. Introduction

Robotic-assisted surgery is when a robot is specifically designed to aid a surgeon in the operating room. A controversially debated issue today is how heavily involved these robots should interact with the surgeon and the patient during surgery. By understanding how surgical robots work, the ethics of robotic-assisted surgery can become more clear. This article will go in-depth on the different types of sensors, how they work, and their advantages and disadvantages.

II. The Relationship of Robots and Surgeons

When discussing sensors in robot-assisted surgery, it is necessary to discuss how controversial the topic remains. There are many advantages that come along with robot-assisted surgery, such as less risk of complications, shorter recovery time, and many other benefits [1]. However, there are complications that come along with these advantages.

The largest disadvantage in robot-assisted surgery is the surgeon's inability to use their natural senses to perform the surgery, especially when it comes to the more complex robots. Due to the nature of robotic surgery, surgeons have less ability to sense their surroundings, which causes an overcompensation of force and potentially injuring the patient [2]. One study by Rodrigues found that more experienced surgeons used more force using a robot than if they had done the surgery without assistance from a robot [3]. Another study was conducted by Meccariello to see if haptics were a necessary component for robotic surgery. This study was conducted to determine if visual feedback is a suitable substitute or if tactile feedback is a necessity for robot-assisted surgery. The study found that, similar to the other studies, tactile feedback was desirable [4].

Costs are often an issue that arises when developing new products, especially for robots and sensors for robot-assisted surgeries. Ng conducted a study in the United States and found that robot-assisted surgery is more expensive than traditional surgery and that this cost margin has widened over time [5]. These higher costs are partially due to their cost of acquisition and maintenance [6].

III. Types of Force and Torque Sensors

For a successful robotic-assisted surgery, the robot must contain a variety of sensors. The most common, and arguably the most important sensors used in surgery robots, are the force and torque sensors. These sensors allow the robot to know where it is and how much force is being applied to the patient. The main types of sensors used for force and torque sensing in robotic surgery are strain gauges, linear variable differential transformers (LVDT), capacitive sensors, and fiber optic strains.

A. Strain Gauges

A strain gauge measures the resistance applied to a load cell while turning. By attaching a secondary load cell, the strain gauge can be used for torque sensing. This allows the strain gauge to act accurately, precisely, and quickly when sensing force and torque.

Two of the largest issues with this method are installation and noise. Adding these sensors adds significant complexity to both the system and the electrical wiring upon set-up. Other downsides of strain gauges are that they tend to be expensive and fragile. Most of the cost is said to be due to the process of fabrication; this process is not well automated due to the complex nature of a metallic structure [7][8].

The most concerning issue in strain gauges is the torque ripple. When the sinusoidal waves interact to form a harmonic drive in a strain gauge they, in turn, deform the flex-spline due to their elliptical shape [7]. This produces unwanted information for the torque sensors. A successful study by Lu created an algorithm to create a feed-forward disturbance observer that can aid in correction and compensation for error [9].

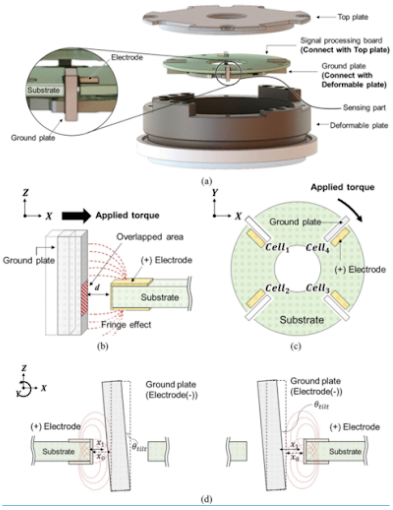


Fig. 1: Strain Gauge Diagram

B. Linear Variable Differential Transformers

It could be considered deceiving to say that a Linear Variable Differential Transformer (LVDT) is a force and torque sensor as these sensors directly measure position and, thus, calculate the force and torque [1]. These devices are not common within the context of force and torque sensing. The design used by Kang found that force accuracy could be calculated within 3.5% and torque within 7.5% [10]. Despite the promising outlook in Kang's study, the research for this type of force and torque seems to have halted. This could be due to the low accuracy of force and torque measuring sensors compared to other types of force and torque sensors.

Fig. 2: LVDT Sensor

C. Capacitive Sensors

Capacitive sensors work by measuring the capacitance between two conductive electrodes using a dielectric material in-between them. This specific "sandwiched" structure allows the force to be measured through the change in capacitance. There are two ways in which a force can be applied to a capacitive sensor: shear and normal. A normal force is a force that is applied perpendicular to the top of the sensor whereas a shear force is applied parallel to the top of the sensor.

Although it seems to be a rather unpopular choice, the capacitive sensors work well as a force and torque sensor. Kim found, in their study, that using a capacitive sensor provides high-sensitivity force and torque sensing. The study further found that these types of sensors are much more feasible in terms of manufacturing. Unfortunately, these sensors are not precise. The relative error and inconsistency can be high when using these types of force and torque sensors, which is not suitable for medical practices [11].

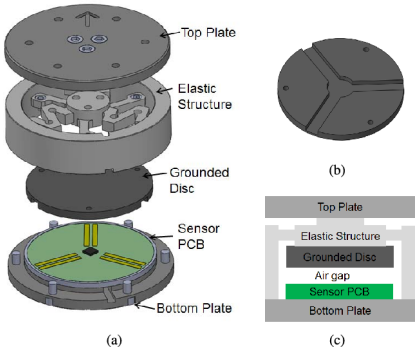


Fig. 3: Capacitive Sensor [8]

D. Optical Fiber

One of the most common implementations of force and torque sensors are those that use optical fiber measurement methods. Optical fibers send a light source, usually using a light-emitting diode (LED), and receive this signal through an optical encoder. Optical sensors are desirable for many reasons: small size, immunity to electromagnetic interference, resistance to water damage, and their ability to be used as a force and torque sensor. Although this is increasing in popularity and has promising results, they have not been as popular to be adopted into practice. This is partially due to their price and their sensitive nature.

These sensors are not perfect. Fiber optic cables can be expensive to produce due to the precise way in which they function. In the same vein, optical fiber cables are very sensitive and can be damaged easily and unpredictably [1][12].

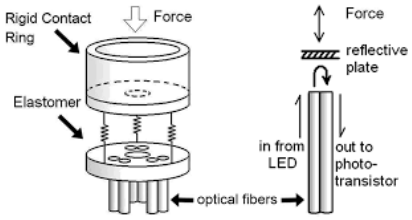


Fig. 4: Optical Fiber Sensor [2]

IV. Axial Force Sensors

Degrees of freedom (DoF) allows for multiple spatial directions to be measured and detected [12]. Depending on their intended use, force and torque sensors can have a large range of DoF. A force and torque sensor's number of axes is equivalent to its number of DoF. Most sensors will either have three DoF or six DoF that allow for greater mobility [13]. However, a larger quantity of DoF is not always linked to a better design, as some DoF could be considered redundant. Doing so would unnecessarily increase the complexity of the design. In spite of designs for robot-assisted surgery with one and two DoF, these sensors are not common.

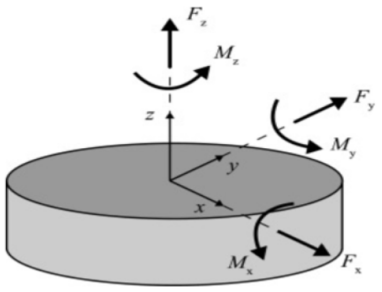


Fig. 5: Cartesian Coordinate System [12]

A. One and Two Degree(s) of Freedom

When a sensor has one or two DoF, it is usually because the sensor is designed for a specific use. A study done by Yip showed a uniaxial force sensor which was designed for surgery involving a beating heart [2]. While using only one DoF allows for a simpler design, complications can arise due to a lack of information provided by the sensor. In Hong and Jo's study, a set of forceps (a tool used during surgery) was designed with two DoF. This allowed for both a compact design and the ability to measure forces [13].

B. Three Degrees of Freedom

A sensor containing three DoF is the most common type of sensors. This is because it can better represent 3D spaces. The most common three-DoF force sensor is a specific type of sensor called a "crossbeam" (or "Maltese cross") type [12], as illustrated in Figure 6.

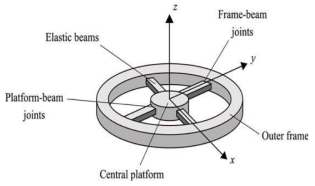


Fig. 6: Cross Beam Force Sensor [12]

According to Templeman, there are three major combinations of DoF chosen for triaxial sensor designs. The first of these three (C1) are Fx, Fy, and Fz, which are applied when there is minor bending required for the z-axis. This design commonly uses Fx and Fy offset to account for bending outside of the resting state orientation. The second combination (C2) is very similar to the first, design-wise. Instead of choosing Fx, Fy, and Fz, the combination of Fz, Mx, and My are chosen. The last combination (C3) includes Fx, Fy, and Mz and is used when there is little to no bending perpendicular to the crossbeam.

C. Six Degrees of Freedom

Force sensors with six DoF are the most common kind of force sensors. As seen in Figure 5, a sensor with six DoF can provide information for all six axes depicted. However, Templeman illustrates how there are many different categories and implementations of the crossbeam structure. The different classifications of the six-DoF crossbeam sensors listed are rigid jointed, flexible jointed, and dual layer. Templeman references other categories for six DoF including column, beam-column, and Stewart platform [12].

V. Conclusion

The discussion on robot-assisted surgery considers how the surgeon interacts with the robot, which sensor is most suitable to the job, and how to streamline the physics of each sensor. Despite the controversies and the complexity, robot-assisted surgery is a promising field that could drastically improve the quality of life for many people in the health-industry.

References

[1] P. Puangmali et al. ""State-of-the-art in force and tactile sensing for minimally invasive surgery"". In: *IEEE Sensors Journal* 8.4 (2008). DOI: 10.1109/jsen.2008.917481.

[2] M. C. Yip, S. G. Yuen, and Howe R. D. ""A robust uniaxial force sensor for minimally invasive surgery"". In: *IEEE Transactions on Biomedical Engineering* 57.5 (2010). DOI: 10.1109/tbme.2009.2039570.

[3] S. P. Rodrigues et al. ""Tying different knots: What forces do we use?"". In: *Surgical Endoscopy* 29.7 (2014). DOI: 10.1007/s00464-014-3898-7.

[4] G. Meccariello and et al. ""An experimental study about haptic feedback in robotic surgery: May visual feedback substitute tactile feedback?"". In: *Journal of Robotic Surgery* 10.1 (2015). DOI: 10.1007/s11701-015-0541-0.

[5] A. P. Ng and et al. ""National analysis of cost disparities in robotic-assisted versus laparoscopic abdominal operations"". In: *Surgery* 173.6 (2023). DOI: 10.1016/j.surg.2023.02.016.

[6] I. Gkegkes, I. Mamais, and C. Iavazzo. ""Robotics in general surgery: A systematic cost assessment"". In: *Journal of Minimal Access Surgery* 13.4 (2017). DOI: 10.4103/0972-9941.195565.

[7] N. Kashiri, J. Malzahn, and N. G. Tsagarakis. ""On the sensor design of torque controlled actuators: A comparison study of strain gauge and encoder-based principles"". In: *IEEE Robotics and Automation Letters* 2.2 (2017). DOI: 10.1109/lra.2017.2662744.

[8] D.-H. Lee et al. ""A capacitive-type novel six-axis force/torque sensor for Robotic Applications"". In: *IEEE Sensors Journal* 16.8 (2016). DOI: 10.1109/jsen.2015.2504267.

[9] Y.-S. Lu et al. ""A torque-ripple compensation scheme for Harmonic Drive Systems"". In: *Electrical Engineering* 95.4 (2012). DOI: 10.1007/978-3-642-33932-5_59.

[10] C.-G. Kang. ""Closed-form force sensing of a 6-axis force transducer based on the stewart platform"". In: *Sensors and Actuators A: Physical* 90.1–2 (2001). DOI: 10.1016/s0924-4247(00)00564-1.

[11] U. Kim et al. ""A novel Six-axis force/torque sensor for Robotic Applications"". In: *IEEE/ASME Transactions on Mechatronics* 22.3 (2017). DOI: 10.1109/tmech.2016.2640194.

[12] J. O. Templeman, B. B. Sheil, and T. Sun. ""Multi-axis force sensors: A state-of-the-art review"". In: *Sensors and Actuators A: Physical* 304 (2020). DOI: 10.1016/j.sna.2019.111772.

[13] M. B. Hong and Y.-H. Jo. ""Design and evaluation of 2-DOF compliant forceps with Force-sensing capability for Minimally Invasive Robot surgery"". In: *IEEE Transactions on Robotics* 28.4 (2012). DOI: 10.1109/tro.2012.2194889.

Unveiling Student Attitudes: Investigating the Influence of Beliefs and Backgrounds on Generative AI Tool Usage

Phan Anh Duong, *BS, Comp. Sci.*

***Abstract*—Generative Artificial Intelligence has had an irreversible effect on the world. AI tools, such as ChatGPT, can be used for a myriad of cases, and has transformed the academic, professional, and personal lives of many. This article will focus on the role of AI as seen and used by university students. We will highlight the beliefs and opinions of a diverse range of individuals about this topic. Through this, we aim to explore the nuances of Generative AI perception and how it might shape future society.**

I. Introduction

Since the beginning of 2023, there has been a rapid rise in public concern of Artificial Intelligence (hereafter AI) in general [1] and Generative Artificial Intelligence (hereafter GenAI) tools in specific. In general terms, GenAI encompass a myriad of systems powered by machine learning algorithms that are capable of creation tasks untypical of a program, such as generating novel texts, images and music. Although GenAI algorithms were already circulating the public before, global attention on such tools markedly increased after the launch of ChatGPT by OpenAI in late 2022. Powered by an upgraded version of OpenAI's previously released text generation model GPT-3, ChatGPT is vastly knowledgeable and capable of taking previous context in a conversation as input for a more accurate and meaningful reply, making users feel like they were "chatting" with the model. This spark of public interest in AI-powered tools has subsequently lead to rapid development and release of other powerful models to the public, such as DALL-E, Stable Diffusion, Midjourney, Bard and BingAI. Their ability to handle complex prompts and produce human-like output has lead to interests into the integration of GenAI in various fields such as healthcare, medicine, education, media, and tourism [2].

This has had many adverse and unforeseeable impacts on the field of education. Open-access release of the aforementioned GenAI tools majorly disrupted teaching and learning in universities worldwide due to their model's ability to generate content that can pass traditional university assessments [3]. Due to this effect, some educators now favor adapted assessments that assume AI usage to encourage critical thinking [4]. Student reactions are more mixed, with some concerned about loss of creativity [4] while others are worried about the reduction of student-teacher relationships [5]. It is clear that in order to further act on the aforementioned disruption in academic institutions caused by GenAI, this disparity between educators and students must be investigated and addressed.

There is a considerable amount of literature on the attitudes and perspectives of educators and students on AI, but this paper will primarily focus on students and the literature in that sector. In their analysis of 399 Hong Kong university students' perceptions of GenAI, Cecilia and co-workers concluded that it was evident that these students are generally familiar with the technologies, showing a good understanding of the capabilities and limitations, as well as a positive attitude towards usage of GenAI in their daily lives. The same positive perception is shared with the second and third year Romanian students examined in another study [5]. A subset of the student population - Canadian health students - were also examined and were found to be cautiously optimistic about the role of AI in their field, although they mostly felt uneducated on the topic [6], unlike the Hong Kong students mentioned above. Other papers [4, 7] highlight the mixed and conflicted beliefs of students on the potential impacts of AI, noting that there exist a tension between perceived benefits and concern on AI's negative effects. Despite these conflicting and diverse findings on student attitudes, some studies [3, 2] share a consensus that students' confidence and familiarity level in GenAI usage is affected by their experience and knowledge in the topic. However, to the best of our knowledge, few researchers have investigated the categorical details of GenAI tool usage in students and how it might be affected by their outlook on AI, instead focusing on student attitudes and perspectives on the impact of such tools instead. This presents an opportunity to conduct a study that seek to categorize a student's categorical usage of Generative AI tools and attempt to link the usage patterns to their personal background.

The aim of this study is to examine the following questions:

1) What are the prominent ways in which university students utilize Generative Artificial Intelligence tools in their daily life and in the academic scene?
2) What is the relationship between a student's beliefs, their backgrounds and their usage pattern of Generative AI?

II. Methods

A. Participants

The study surveyed 111 students from diverse global universities via an online questionnaire. Participants, aged 17 to 24 (Average = 20), represented a mix of academic levels, and hailed from various global locations. The surveyed students demonstrated diverse academic interests, with 55 majoring in STEM fields, 21 in Arts and Humanities, 10 in Social Sciences, 12 in Business, 6 in Public and Social Services, 2 in Health and Medicine, and 5 in Multi-Disciplinary Fields,

encompassing majors that span multiple disciplines beyond the specified categories.

B. Data Collection Instrument

The main instrument for data collection in this study was an online questionnaire via Google Form. The survey consisted of four parts, with one reserved for international students. Part 2 of the survey, which aimed to capture students' beliefs on Generative AI was adapted from a similar study conducted on UK medical students [8]. The other parts of the survey were designed by the author. The following images display the results of the survey grouped into categories of majors.

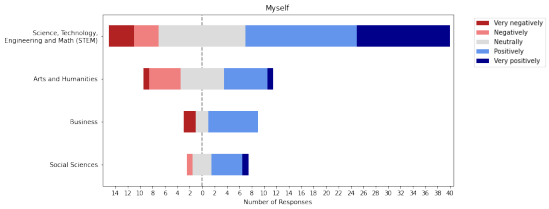


Fig. 1: An overview of the positive and negative views concerning personal beliefs and competencies regarding AI.

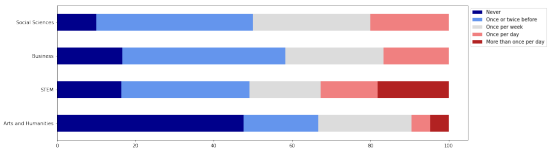


Fig. 2: Information regarding the personal use of ChatGPT.

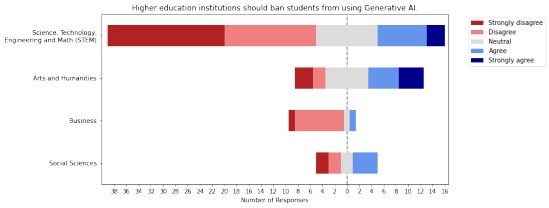


Fig. 3: Positive and negative views grouped into types of majors on banning Generative AI usage in higher education institutions.

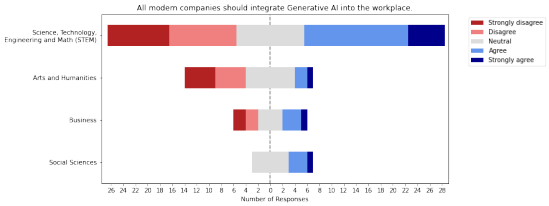


Fig. 4: Positive and negative views grouped into types of majors on integrating Generative AI in modern companies.

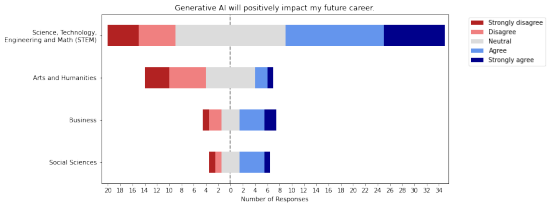


Fig. 5: Positive and negative views grouped into types of majors regarding the personal, positive impact of Generative AI on participants' careers.

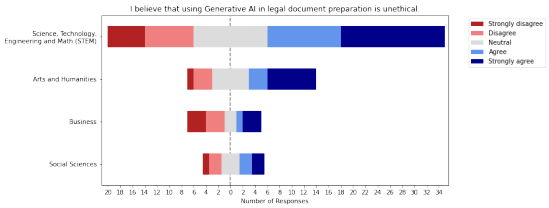


Fig. 6: Positive and negative views grouped into types of majors on ethically using Generative AI in legal document preparation.

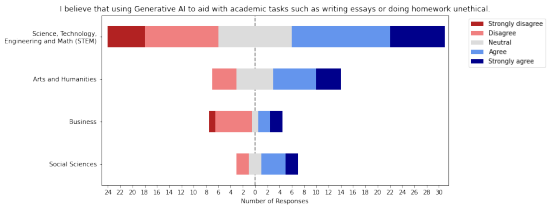


Fig. 7: Positive and negative views grouped into types of majors on ethically using Generative AI in academia.

III. Discussion

This study endeavors to examine the personal perspectives and ethical considerations of university students regarding Generative AI, along with investigating their usage patterns and adoption of Generative AI technologies. Our survey, encompassing 111 students from diverse disciplines and programs worldwide, facilitated a comprehensive examination of these aspects through a questionnaire. Broadly, the findings reveal that a majority of students are familiar with Generative AI technologies, and are generally positive towards such tools. Concurrently, there is a consensus among students for the implementation of more stringent laws and regulations, coupled with enhanced educational efforts surrounding the technology.

The inquiry into the predominant ways in which students integrate Generative AI tools into their lives has yielded illuminating insights. Significantly, a noteworthy proportion of surveyed students disclosed their prior utilization of Generative AI tools, engaging with them across diverse applications, ranging from leisurely self-entertainment to assistance in academic tasks. Despite the diversity of applications, it is noteworthy that the preeminent Generative AI tool employed by students is ChatGPT—a text generation model. This underscores the adaptability and practicality of text-based Generative AI tools,

emerging as the predominant category among student users.

The observed trend of students incorporating Generative AI tools into their learning and research endeavors aligns with existing literature [2], substantiating the prevailing positive disposition of university students towards leveraging Generative AI technologies for personal enrichment.

Predominantly, students employed Generative AI tools for recreational purposes, indicating a perception of Generative AI as an avenue for entertainment, transcending its professional and technical applications. This inclination suggests that Generative AI tools inherently possess engaging attributes that resonate with students. Nevertheless, the second most prevalent use case for Generative AI is its application in academic work. The discerned reliance on Generative AI tools for academic tasks signals an evolving acknowledgment of their potential to enhance learning quality and broaden educational experiences, as elucidated in the work of Mauer Idroes et al. (2023). However, this trend raises apprehensions concerning academic integrity and potential detriments to students' learning due to over-dependence. The third most frequent use case revolves around learning and skill development, aligning with the aforementioned educational utility. This underscores Generative AI's capacity to facilitate skill acquisition and knowledge attainment beyond conventional educational frameworks, eliminating the necessity for direct teacher involvement.

More creative applications of Generative AI, particularly in domains such as art, design, music, and sound generation, remain relatively constrained within the student demographic. This observation aligns with the restricted number of students who reported engaging with image-based and sound-based Generative AI systems in the survey. The limited utilization of Generative AI in creative contexts suggests that students, as of the present study, may not fully exploit the potential of these technologies for artistic and expressive purposes. This restraint could be attributed to factors such as awareness, accessibility, or the perceived ethicality of Generative AI in creative endeavors.

The exploration of the intricate relationship between students' beliefs, disciplinary backgrounds, and utilization patterns of Generative AI has provided valuable insights into the multifaceted dynamics that influence technology adoption. Examination of survey responses revealed a profound connection between a student's academic discipline, their perspectives on Generative AI, and their active engagement with such tools. Specifically, students majoring in Arts & Humanities demonstrated a lower frequency of usage compared to the baseline student population. Furthermore, these students tended to harbor more pessimistic views regarding Generative AI, evident in their dissenting opinions on the integration of Generative AI into the workplace, apprehensions about its potential negative impact on their future careers, and mixed perspectives on their outlook toward Generative AI.

Remarkably, despite expressing reservations about the ethical implications of various Generative AI use cases (excluding personal learning and video game development), a noteworthy proportion of students maintained a positive outlook on Generative AI and continued to use these tools frequently. This apparent paradox suggests a nuanced interplay between ethical considerations and the heightened utilization of Generative AI for personal learning and entertainment. It also hints at the potential prioritization of personal benefits over ethical reservations, as manifested in the discrepancy between students perceiving the use of Generative AI to assist in academic tasks as unethical, and the observation that it currently ranks as the second most popular use case.

This discrepancy can also be examined in another light: a substantial number of students harbor the perception that academic institutions view Generative AI negatively, while conversely, an equally significant proportion of students believe that companies hold a positive view of Generative AI. The perception of academic institutions' negative stance towards Generative AI may foster a sense of divergence or rebellion, prompting students to explore and adopt these technologies out of curiosity or as a form of resistance. On the other hand, the positive perception of Generative AI by companies may contribute to students' motivation to align themselves with the perceived industry endorsement of these tools, potentially enhancing their employability and professional prospects. This observed dichotomy between institutional views and corporate perspectives could also contribute to the complexity of students' ethical considerations. Students may prioritize the perceived endorsement of Generative AI by companies, potentially overlooking or downplaying ethical concerns in favor of aligning with prevailing industry trends. This prioritization of professional and practical considerations over personal ethical reservations could elucidate the apparent inconsistency between students' acknowledgment of ethical concerns and their frequent and varied usage of Generative AI tools. This observation finds empirical support in the survey data, where a predominant majority of students express the belief that Generative AI will have a positive impact on their future careers. This optimistic outlook aligns with the discernible trend of widespread adoption and frequent usage of Generative AI tools, even in the presence of ethical reservations. Notably, the subset of students majoring in Arts & Humanities exhibits a contrasting perspective, predominantly disagreeing with the notion that Generative AI will positively influence their careers. This dissenting view is further manifested in their comparatively lower frequency of engagement with Generative AI tools.

The correlation between positive career perceptions and increased adoption of Generative AI underscores the significant role of career expectations in motivating students to integrate these technologies into their academic and personal spheres. This alignment further emphasizes the intricate relationship between perceived professional benefits, disciplinary perspectives, and ethical considerations in shaping students' attitudes and behaviors toward Generative AI.

Several recent studies [2, 3] propose a connection between a student's familiarity with Generative AI, their knowledge of the subject, and the frequency of usage. This implies a correlation wherein students who frequently engage with Generative AI tools tend to perceive themselves as more competent in the subject matter. Importantly, our data aligns consistently with this implication.

The findings of our study underscore the prevalence of Generative AI usage among students, particularly in academically related tasks. As a substantial number of students already incorporate Generative AI tools into their learning processes and express disagreement with the notion of banning such technologies in academic institutions, schools must consider integrating Generative AI into the educational framework.

One noteworthy avenue for exploration is the incorporation of Generative AI into assignments and exams. By being mindful of Generative AI technologies in the design of academic tasks, institutions can foster an environment that encourages creative and critical thinking. This integration has the potential to empower students in their educational journey, providing them with tools that augment their problem-solving capabilities and enhance their learning experiences.

However, a cautious approach is advised. While the integration of Generative AI presents opportunities for innovation, transparency and effective communication with the student body is paramount. It is essential to address potential concerns, as highlighted in prior studies that emphasize students' apprehensions about the loss of creativity or specific program needs when Generative AI is introduced into educational processes [4, 6]. In navigating this integration, academic institutions should actively involve students in the decision-making process. Seeking student input, addressing concerns, and providing clear information about the purpose and benefits of incorporating Generative AI can contribute to a more positive reception among the student body.

Due to the time limitation of the project, the present study has only managed to capture a disproportional distribution of students in terms of nationality and discipline: the majority of students majored in STEM and hailed from the U.S. The author has only analyzed groups of majors that exceed or meet the critical mass of 10 students, but a more equal distribution of disciplines would allow for better generalization and comparison of different subsets of the study body. Moreover, the limitations of the data collection instrument did not allow for students to elaborate on questions that deserve more nuance, chiefly the ethics and perspective section. In further studies, deeper exploration into the ethical concerns of students is warranted, as it might yield important insights that could explain the discrepancy between a student's ethical reservations and their employment of Generative AI.

REFERENCES

[1] Alec Tyson and Emma Kikuchi. *Growing public concern about the role of artificial intelligence in daily life*. Pew Research Center, Aug. 2023. URL: https://pewrsr.ch/3QZ6H6D (visited on 09/15/2023).

[2] Chan Cecilia and Wenjie Hu. "Students' Voices on Generative AI: Perceptions, Benefits, and Challenges in Higher Education". In: *arXiv preprint arXiv:2305.00290* (Apr. 2023). DOI: 10.48550/arxiv.2305.00290.

[3] Andrew Kelly, Miriam Sullivan, and Katrina Strampel. "Generative artificial intelligence: University student awareness, experience, and confidence in use across disciplines". In: *Journal of University Teaching & Learning Practice* 20 (Aug. 2023). DOI: 10.53761/1.20.6.12. URL: https://ro.uow.edu.au/jutlp/vol20/iss6/12/ (visited on 08/24/2023).

[4] Adele Smolansky et al. "Educator and Student Perspectives on the Impact of Generative AI on Assessments in Higher Education". In: *Proceedings of the Tenth ACM Conference on Learning @ Scale* (July 2023), pp. 378–382. DOI: 10.1145/3573051.3596191. (Visited on 08/24/2023).

[5] Ghazi Mauer Idroes et al. "Student Perspectives on the Role of Artificial Intelligence in Education: A Survey-Based Analysis". In: *Journal of Education Management and Learning* 1 (July 2023), p. 2023. DOI: https://doi.org/10.60084/jeml.v1i1.58. (Visited on 08/24/2023).

[6] Minnie Teng et al. "Health Care Students' Perspectives on Artificial Intelligence: Countrywide Survey in Canada". In: *JMIR Medical Education* 8 (Jan. 2022), e33390. DOI: 10.2196/33390. URL: https://mededu.jmir.org/2022/1/e33390.

[7] Thomas R. Jeffrey. "Understanding Generation Z Perceptions of Artificial Intelligence in Marketing and Advertising". In: *Advertising & Society Quarterly* 22 (2021). DOI: 10.1353/asr.2021.0052.

[8] Cherry Sit et al. "Attitudes and perceptions of UK medical students towards artificial intelligence and radiology: a multicentre survey". In: *Insights into Imaging* 11 (2020). DOI: 10.1186/s13244-019-0830-7. URL: https://eds.b.ebscohost.com/eds/detail/detail?vid=0&sid=585e6bb3-e790-4bc4-8bd3-3895340cc89e%40pdc-v-sessmgr02&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=edssjs.17E189C1&db=edssjs.

Defining the Identity of a Programming Language

Jaran Chao, *B.S., Comp. Sci.,*

***Abstract*—This work is an addendum and continuation to an article published in the IEEE Student Magazine Volume 3. This work will discuss how language features affect the identity of a programming language and amend this understanding with programming paradigms. Typically, programming languages are designed for general purpose programming. Some languages more well-equipped to build specific applications than others. Furthermore, some languages, known as domain-specific languages, or DSLs, are specifically built for one application domain. For the purpose of this work, we will focus on the effects on general-purpose programming language identities. However, many of the covered topics can be similarly applied to DSLs.**

I. Introduction

New programming languages are born every day to solve new problems. Understanding these complex languages requires us to understand each one's programming paradigms, styles, and features. However, programming languages are quite complex and many factors contribute to giving a language its unique identity. While the previous article focused on core design choices summarized in preliminary concepts, this work will focus on high level design and implementation details.

A. Comparing Programming Languages

Programming languages are very complex works of engineering. As such, this work will focus on two factors on a programming language's identity: *programming paradigms* and *language features*.

Programming paradigms classify programming languages based on their features. They are concerned with many aspects of a programming language. Some may be concerned with the execution model, others with code organization, and others with the language's syntax and grammar. Furthermore, languages can be classified into multiple paradigms and are often a result of the philosophy and ideology behind a language's design and implementation.

The feature set of a language will concern the semantics that allows programmers to convey complex ideas into programs. Language features can affect many aspects of the language, from grammar and syntax to ideology and philosophy. Furthermore, the amount of language features is vast. Therefore, this report will focus on the three following feature sets:

1) Features defined and tied to certain Programming Paradigms.
2) Error handling strategies.
3) Concurrency and Parallelism Support.

II. Misconceptions

Before we tackle the differences in programming languages, let us define a programming language and clear up some misconceptions. A programming language is a form of textual or graphical notation for creating software applications. Most programming languages are designed to be general-purpose and can be used to build many applications. Some general-purpose languages are better equipped for specific applications such as low-level/embedded systems, web development, and machine learning. A common misconception is that these general-purpose languages are known as *domain-specific languages*, or DSLs. That is not the case. DSLs are languages built for use solely in a specific application domain and, as such, are not for general-purpose use. Another common misconception is that all DSLs are created equally. There are, in fact, two different types of DSLs: "internal/embedded" DSLs and "external" DSLs.

Internal/Embedded DSLs are created using host language features and rely on the flexibility of the host language to determine their identity. These kinds of DSLs often inherit generic language constructs from their host language and provide domain-specific primitives that allow programmers to work at a higher level of abstraction. Due to this, internal/embedded DSLs often inherit, at least partially, the identity of their host language. External DSLs are created with a separate compiler/interpreter and/or development tooling, similar to general-purpose languages. Therefore, external DSLs are free to build a separate identity from the host language/language of implementation. The final common misconception is found in the terminology surrounding internal/embedded DSLs. Both terms, "internal" and "embedded", describe a DSL that is created inside another language using the host language's facilities to create both syntax and semantics. The term "embedded" is more common in literature surrounding DSLs. However, some experts believe that "embedded" did not properly convey this definition and instead implied something entirely different. "Internal"/"external" terminology was created, and possibly initially coined by Martin Fowler[1], to address this distinction.

For the purpose of this work, we will focus on the identities of general-purpose programming languages. However, many of the topics covered can be similarly applied to external DSLs and, to some extent, internal/embedded DSLs.

III. Preliminary Concepts

This section serves as a brief summary and further description/analysis of language properties for an article published in the IEEE Student Magazine Volume 3. These language properties include: Compilation vs Interpretation, Compilation Targets, and Type Systems.

A. Compilation vs. Interpretation

The first significant distinction between languages is how the resulting written program executes. *Compiled* and *interpreted* are the two main types of programming language. As defined in *Crafting interpreters*[2] by Nystrom:

- A language implementation "is compiled" or "is a compiler" when it translates source code to some–usually lower-level–form and does not execute it. The user has to take the resulting output and run it themselves. If the language generates machine code or bytecode, it is compiling. If the language generates another high-level language, also known as transpilation, it is compiling.
- A language implementation "is interpreted" or "is an interpreter" when it takes in source code and executes it immediately. It runs the program "from source".

As seen below, this distinction is not purely black and white. In practice, most modern languages utilize both compilers and interpreters in the build/execution process. The identity of the language is associated with the user experience of using the language, not the implementation.

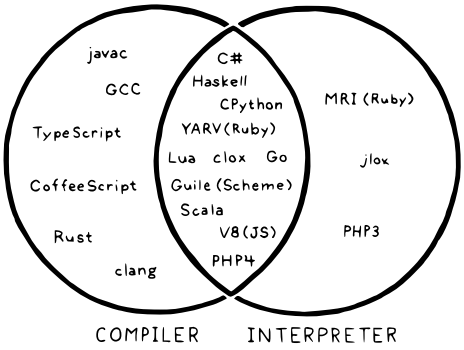


Fig. 1: Venn diagram of compiled languages vs interpreted languages[2].

B. Compilation Targets

The second significant distinction between languages is where the resulting written program executes. There are three main structures: compiling to a native binary, tree-walking interpretation, and compiling to an intermediate representation. Each structure has its own capabilities, advantages, and disadvantages.

1) Compiling to native binary: Native binaries, or native executables, are a common compilation target that consist of machine code. This code is understood by the underlying operating and CPU and, due to a lack of overhead, no additional software of translation units are required. Furthermore, the compiler can make fine-tuned and architecture-specific optimizations per platform.

Unfortunately, this precision comes at a cost. Not every platform runs using the same machine code because every CPU has its own instruction set. This means that a program cannot run on another OS or CPU without being recompiled for that architecture and hindering the ability to compile across platforms. A new compiler backend must be created for every new platform that outputs valid machine code. This is tedious work and can have dire consequences on a language's identity. Users expect a language to work on their platform. Missing a specific platform signals the immaturity of a language.

Languages can circumvent this issue by choosing a target language to compile to. The most common are C and LLVM IR, however, any language that compiles to native binaries can be utilized. Instead of writing and maintaining a compiler for multiple platforms, languages can utilize existing compilation tools and tap into the hundreds of thousands of engineering hours put into said tools for optimization and code generation. This allows languages to have excellent cross-platform capabilities, however, this may affect the language's identity as the underlying target language can have ripple effects and affect other design decisions and/or the user's experience.

2) Tree-Walking Interpreters: Tree-walking interpreters are quick to implement because they traverse the generated syntax tree after parsing, directly interpreting the source code syntax tree. However, this approach is very slow and resource intensive because of how syntax trees are stored in memory. This leads to pointer chasing and high memory usage. As such, tree-walking interpreters significantly impact the identity of a language in terms of performance. These aspects make tree-walking interpreters good for external DSLs in non-performance-intensive tasks or rapid prototyping, however, many mainstream industrial languages today do not take this approach.

3) Compiling to Intermediate Representation: Intermediate Representation, or IR, is a scalable data structure used internally by compilers and interpreters to represent source code. IR usually takes the form of a *bytecode* or a *graph data structure* and is designed to be expressive and conducive to further manipulation, such as optimizations or translations[3]. Due to its scalability, new platforms can be easily added as the language VM/code generator works on the lowered IR, which is not as complex as the high-level language source code it represents. However, the main downside to this approach is performance. The overhead of the language VM can significantly decrease performance in some cases and, due to practical constraints, the language's scalability is not as pronounced. This can negatively impact a language's identity as being slow. The impact of this approach to language identity can vary greatly depending on implementation.

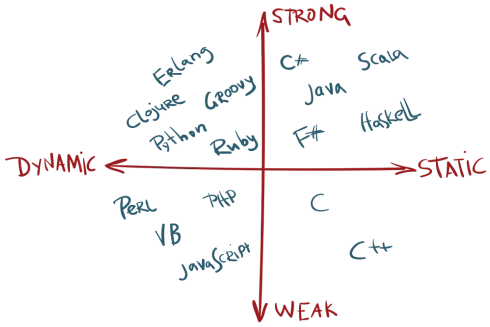


Fig. 2: Four quadrant graph showing whether a language is statically or dynamically-typed. As well as whether a language is strongly or weakly typed[4].

C. Type Systems

The final significant distinction addressed in the previous article is the language's type system, one of the primary bases for the rest of the features and identity of a language. Type systems have two central opposites: *Strong vs. Weak* and *Static vs. Dynamic*. Most languages, with a few exceptions like Forth, have some form of a type system that can be visualized as a 2D plane, as seen in Figure 2.

For this work, static and dynamic typing will be defined as the following:

Static Typing: the type-checking process knows the type of a variable or function ahead of time. This process usually occurs during compilation and, as such, statically-typed languages have an intermediary step that performs type-checking during compilation.

Dynamic Typing: the type-checking process does not know the type of a variable or function ahead of time and is deferred to runtime. Therefore, dynamically-typed languages are often associated with a runtime, allowing the retention of type information to perform this type-checking process.

While static and dynamic typing have concrete definitions by the programming language community, strong and weak typing do not. The definitions are fluid and may lead to confusion. Ovid states that the most common classification for type systems is "strong" or "weak". It is unfortunate that these words have nearly no meaning at all[5, para. 4]. For those reasons, this work will not discuss strong vs. weak typing.

1) Type Inference: The previous article only mentions type inference by name, leaving it as an exercise for the reader to ponder further. In contrast, this work will briefly discuss type inference. To clarify, type inference is a process a compiler performs to infer the type annotations of a program from the source code. This process was commonly seen in functional languages (discussed later) such as Haskell, OCaml, and F# as these languages employed type inference through the Hindley-Milner type system, also known as HM, and its derivatives. HM type systems allow for full type inference of a program without needing any type definitions, first described by Hindley in 1969[6] and rediscovered by Milner in 1978[7].

Other languages have gradually adopted type inference into their design. Languages like C++, Java, Kotlin, Scala, Swift, and more have some form of type inference capabilities. Furthermore, type inference allows developers to have the development experience of a dynamically-typed language while not sacrificing the safety of a statically-typed language, overall positively affecting the identity of a language. However, due to type inference's algorithmic complexity, compilation times are greatly increased because the compiler must validate the program with minimal information, negatively affecting a language's identity.

IV. Programming Paradigms

The identity of a language is often discussed in relation to the paradigms the language supports. This work will focus on the most commonly discussed programming paradigms: *imperative* and *declarative*. Many modern languages are multi-paradigm languages, meaning they support more than one programming paradigm and, therefore, may have many sub-identities in terms of ideology and philosophy that stem from multiple paradigms.

A. What is a Programming Paradigm?

Programming paradigms classify programming languages based on their features. They are concerned with many aspects of a programming language such as the execution model, code organization, and/or a language's syntax and grammar. Furthermore, languages can be classified into multiple paradigms and are often a result of the philosophy and ideology behind a language's design and implementation.

B. Imperative

The imperative programming paradigm manipulates a program's state using instructions that were given to it. Imperative programming focuses on how a program operates step-by-step rather than on a high-level description of its expected results. Procedural programming and object-oriented programming are the two most common paradigms under imperative programming.

1) Procedural: Procedural programming is based on procedures: a collection of computational statements to be executed. Any given procedure could be called at any moment during program execution. Procedures can call other procedures and even call itself. Many languages support procedural programming, the first of which were Fortran, ALGOL, and COBOL. Many modern languages support procedural programming, such as C++, Java, Kotlin, Swift, Scala, Python, and Javascript.

2) Object Oriented: Object-Oriented programming is based on objects: a collection of data, in the form of fields, and code, in the form of methods. The program will manipulate its state through the interactions between objects. When objects respond and send messages to each other, that is referred to as *message passing*. Many programming languages support object-oriented programming; the first were Simula, Smalltalk, and Self. Many modern languages support object-oriented programming such as C++, Java, Kotlin, Swift, Scala, Python, and Javascript.

C. Declarative

Declarative programming is a programming paradigm in which the structure and elements of the program express the computation logic without describing the control flow, focusing on the high level description of its expected results rather than the step-by-step instructions needed to achieve that result. In essence, declarative programming tells the program what computation to do without regard to how that computation is completed. Functional and logic programming are the two most common paradigms under declarative programming.

1) Functional: Functional programming regards functions as first-class citizens. First-class citizens are an entity within a language which supports all generally available operations. This includes being passed as an argument, returned from a

function, and assigned to variables/stored in data structures. For functions to be first-class, these operations must be applicable. These functions are then applied, composed, and passed around to construct the logic behind a program. Instead of a sequence of statements that update the state of the program, a tree of function expressions which maps values to other values. This allows complex abstractions from other paradigms to be modeled as functions. Many programming languages support functional programming; the first were Lisp and ML. Many modern languages support functional programming, such as Haskell, the ML family (ML, Standard ML, OCaml, F#), Kotlin, Swift, Scala, Python, and Javascript.

2) Logic: Logic programming is largely based on *formal logic*. This formal logic is stated in the form of clauses, which express facts and rules about the program's domain. Instead of a sequence of statements or a tree of functions that manipulate the program state, logic programming solves queries of the system based on the rules and facts of the program's domain. Unlike the previously mentioned paradigms, only one language family exists for logic programming: Prolog.

D. Family Trees

Programming languages take influence from their predecessors. Supported programming paradigms are often inherited down these family trees. An example of this can be seen below. ALGOL, Lisp, APL, and Prolog created their own family trees, and most modern languages can trace their history and identity back to one of these languages. Not all family trees start at the top of the ancestry chain. An example of this is Simula, which started the object oriented family tree. This influence is not always through supported programming paradigms, it may contain syntactical or runtime characteristics. This can be seen in languages like Elixir, where a syntactical influence came from Ruby and a runtime influence came from Erlang as Elixir runs on the same platform as Erlang (the BEAM Virtual Machine) and is interoperable with Erlang.

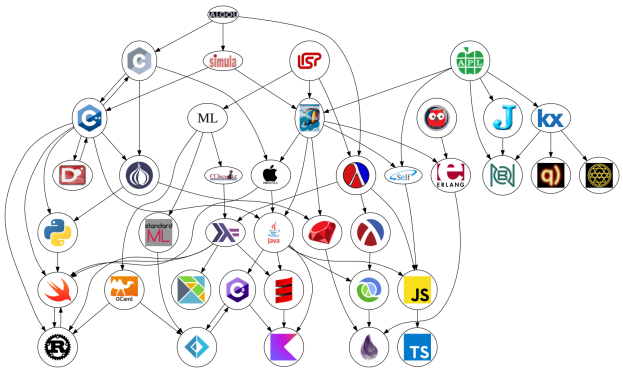


Fig. 3: Graph of programming language family tree, by influence[8].

1) "Family Heads": Family trees usually start from one language; this language will be classified as the "family head" in this work. As mentioned before, ALGOL, Lisp, APL, and Prolog can be seen as a "family head" of their respective programming paradigms. Other languages, like the previously mentioned Simula, can be in the middle the ancestry chain and still be a "family head". The commonly considered "family heads" of each programming paradigm are as follows: C and the lesser known ALGOL for imperative, specifically procedural, programming (C has a large influence on syntax, commonly known as C-style syntax); Lisp and ML for functional programming (functional programming is split into two main families, known as the Lisp family and the ML family. However, there are outliers such as Haskell, Scala, and Elm which are not considered to be in either of the two families); APL for array programming (a paradigm in which operations apply transparently to vectors, matrices, and higher-dimensional arrays/tensors); and Prolog for logic programming.

V. Language Features

The final major distinction of a language's identity is its chosen feature set. While the previous distinctions play a pivotal role in creating the identity of a language, this distinction plays the biggest part in a language's public perception. Language features immediately affect a language's identity in a user's mind, positively if a feature they desire is included, or negatively if a feature is missing. Language features also affect the user's experience. This is particularly seen in feature compatibility or in feature overlap. If two features actively clash against one another, it may adversely affect the language's identity and development experience. Likewise, if multiple features can be used to solve the same problem, this can lead to confusion and disagreement on which feature to use for a particular problem. This will affect the language's identity, but in a more subjective manner that is dependent on the user. There are too many existing language features to cover in this work. For this reason, only three areas of language features, most applicable to real world practical programming, will be discussed.

A. Paradigm Features

Paradigm features are dictated by the language's chosen philosophy and ideology surrounding supported paradigms. Such features are often required for the language's identity to claim that it sufficiently supports the paradigm.

For example, a language must support the concept of procedures, in some form, to support procedural programming. This has been commonly generalized to functions. However, a key difference between procedures and functions is their nature when they end. Functions return a value. Functions will also have a return type in a statically-typed language. Procedures do not return a value. To denote this, statically-typed languages have created a so-called *empty type*. Oftentimes, this value is denoted as void, none/nil, or unit.

Features may also have effects on other parts of a language's identity. Recall that functional programming states that functions are first-class citizens, able to be passed around like data. This means a language must include a function as a type in the type system. For dynamically-typed languages, this may not be an issue as type declarations are not needed, meaning a function type may never be exposed to the user.

However, a statically-typed language would need to expose a function type to be used in type declarations. Incorporating built-in function types into the language's type syntax during development is a simple fix for languages being designed with this feature. However, for languages trying to retroactively add function types to an existing language without breaking backward compatibility, this can often lead to hacky and ad-hoc solutions. An example of this is Java. Java's identity, prior to Java 8, was purely object-oriented. To change its identity to support more functional features, Java had to introduce *Single Abstract Method types* through JEP 126[9]. This ad-hoc solution paved the way for future Java releases to begin adopting more and more functional features, updating its identity.

Finally, the last type of feature dictated by a paradigm is actually dictated by the lack of a paradigm. Recall object-oriented programming and how it bundles state with the code that modifies that state. This programming paradigm surged in popularity with languages such as C++ and Java. These languages are easy to learn and follow by keeping fields and methods in the same location. The object-oriented model also maps nicely to real-world modeling with features such as inheritance and polymorphism. Due to this, languages that do not explicitly support object-oriented programming have incorporated features that allow emulating object-oriented patterns, such as inheritance. For example, Rust and Go both do not explicitly support object-oriented patterns. Yet both Rust[10] and Go[11] have ways to emulate object-oriented patterns.

Overall, the features dictated by paradigms do not contradict the established identity of a language. Instead, they complement and, in some cases, add a new identity to a language.

B. Error Handling Strategies

The world is imperfect, and the unhappy path is often the most traveled. Handling error states at runtime is very important and error handling strategies can dictate a language's identity. Of the many error-handling strategies available, this work will focus on the three common approaches of modern languages: exceptions, errors as values, and algebraic effects. While this work will discuss these three strategies, their advantages, and disadvantages independently, however, it is possible for multiple strategies to exist in a single language.

1) Exceptions: Exceptions are states of a program from which the program cannot recover from and continue execution. Depending on language terminology, exceptions are often thrown or raised when an unrecoverable state has been reached. This immediately halts normal control flow and immediately begins unwinding the program call stack to terminate the program. If the program encounters an exception handler during this stack unwinding process, often denoted as a catch, except, or rescue block, the program will resume normal control flow at the exception handler. If no exception handler is found, the global exception handler is used, leading to program termination.

Exceptions have the advantage of being easily detected. As an exception unwinds the call stack, it maintains a record of visited call stack sections known as *stack frames*. This is called a *stack trace* and it allows developers to find the root causes of exceptions and fix them. Exceptions also have the advantage of being easily handled. Because the program only needs to find an exception handler for the specific exception anywhere in the call stack, an exception handler, many stack frames ago, can still help the program recover from an unrecoverable state.

However, the nature of exceptions can be a disadvantage. Because the program will halt normal control flow at an exception, requiring an exception handler to handle the issue, this is seen as hidden control flow and may lead to issues in debugging. Including exceptions in a language may negatively impact the language's identity, as users may dislike the use of hidden control flow.

2) Errors as Values: Errors as values is an error handling strategy where errors are treated as valid values in a program and can be passed around as normal values until they are handled. This has the advantage of maintaining normal control flow for errors, making it easier to reason about the error-handling logic of a program. However, this propagation of errors to later portions of the program loses out on the ability to find the origin of the error. These error values can show up hundreds of lines away from the source of the error and, due to the propagation of a valid state, the program is none the wiser until it reaches a part of the program that recognizes it as an error state. This can be circumvented if the error handling is placed immediately after the error source. However, errors as values are not easily composable, and if errors need to be propagated and composed with other errors, it can become quite tedious.

One solution is to have an error value representing every permutation of possible errors in a computation chain. This gives the language and the developer a traceable history of the error propagation which, in turn, speeds up debugging, but slows down development time as every permutation of the error must be accounted for. Another solution is to have error values override each other and have the error with the highest "priority" returned and handled. This allows the language and the developer to simply handle each individual error state without the need to combine them. However, this means errors are hidden until the previous error has been fixed, leading to increased development time.

These disadvantages of errors as values may lead to poorer development experiences when using a language that incorporates them as the primary error-handling strategy. A user may subjectively view these disadvantages as negatively impacting the identity of a language, but this depends on each user and use case.

3) Algebraic Effects: Algebraic effects are a general approach to computational effects based on the premise that impure behavior arises from a set of known operations. These operations, known as *effects*, can include: get & set for accessing mutable state, read & print for interactive I/O, or raise/throw for exceptions. This allows for the natural evolution of *handlers* handling exceptions and any other effect, creating one effective strategy to handle a multitude of programming challenges[12].

In more recent years, industrial languages, such as OCaml, have adopted algebraic effects, and efforts have been made to

add them to languages such as Kotlin[13]. Algebraic effects are still a research topic in languages such as Koka[14], Eff[12], and Effekt[15]. Due to their theoretical nature, this work will briefly discuss core aspects below, but more complex semantics will be mentioned by name and not discussed. Such semantics include: *shallow vs deep handlers*, *lexical/static vs dynamic resolution of handlers*, and *single-shot vs multi-shot continuations*.

Algebraic effects, as they pertain to error handling, can be commonly thought of as exceptions with the ability to resume or a "resumable try/catch"[16]. A core semantic behind algebraic effects is the ability to resume program control at the invocation location of the effect. This is known as *resumption* and is made possible by *continuations*. Continuations reify program control state as a data structure that represents a computation at a given point in execution. In algebraic effects, continuations are often exposed to the user either directly or through an API.

Algebraic effects were designed to combat the issues of both exceptions and errors as values. They allow for resumption of program execution when the error state is recoverable, a semantic very difficult to implement in previous strategies. Their algebraic nature allows for easier composability of errors and for the *scoping* of errors as an effect signature can be tracked throughout the source code, either lexically or dynamically.

Due to the theoretical nature of algebraic effects, their impact on a language's identity is unclear. The inherent complexities of algebraic effects may outweigh their benefits. Only time will tell.

C. Concurrent and Parallel Programming Support

Moore's law is reaching its limit. Support for concurrent and parallel programming is becoming increasingly important for a modern language and as Stutter stated: "The free lunch is over." Code can no longer be made faster with simply faster hardware. Code needs to utilize concurrency and parallelism to gain performance[17, para. 1].

Let us first define concurrency and parallelism.

- *Concurrency* is dealing with multiple tasks at the same time, concurrently. Two or more tasks can start, run, and complete over the same period of time. However, this does not ensure that they will be running simultaneously.
- *Parallelism* is running multiple tasks at the same time in parallel. A task is broken up into subtasks that run simultaneously on some hardware, i.e., a multicore CPU or a GPU.

Out of all the concurrency models, this work will focus on the following four models primarily used in industry languages: Processes, Operating System (OS) Threads, Coroutines, and Event Loops. While this work will discuss, in brief detail, these four models independently, modern languages incorporate many of these models simultaneously. Furthermore, some languages like Go (through Goroutines) and Erlang (through the *actor model*) have largely carved out their identity due to their extensive support for concurrent programming.

1) Processes: When using processes as a concurrency model, a language supports concurrency and parallelism. Since the underlying OS manages processes, they can be run in parallel with other processes on other cores. Furthermore, processes are *preemptively scheduled*, meaning that the OS can interrupt any process at any time to let another process run.

A disadvantage of processes is their cost. They exist as an entirely separate process on the machine, meaning a new environment must be set up for them to run. For this reason, languages such as Python utilize multiprocessing as its concurrency model to circumvent the global interpreter lock[18]. Another disadvantage to the process model is sharing memory. Each process is in its own *virtual address space*, meaning it cannot share memory with other processes. The only way to share memory is through *interprocess communication*, which can be ill-suited for some situations.

2) OS Threads: When using OS threads as a concurrency model, a language supports concurrency and parallelism. Since the underlying OS manages each thread, they can be run in parallel with other threads on other cores. However, just like processes, OS threads are preemptively scheduled and expensive to create, though less expensive than processes. Unlike processes, OS threads share the same virtual address space, meaning sharing memory is simpler. However, OS threads suffer from data races and race conditions, especially if shared data is not properly synchronized between threads. This synchronization makes threaded code hard to debug and manage.

3) Coroutines: When using coroutines as a concurrency model, a language supports concurrency but does not support parallelism. Coroutines are managed by the language runtime instead of the OS. Unlike processes and threads, coroutines are cheap to create and *cooperatively scheduled*. Cooperative scheduling means that the system cannot interrupt a coroutine to run another coroutine. Another coroutine can only be scheduled if the previously running coroutine has yielded/suspended. However, coroutines do not support parallelism as they are usually tied to a singular thread. This means multiple coroutines cannot be run in parallel unless on separate backing threads.

a) What is a coroutine?: The definition of a coroutine is not well-defined in today's literature. At its core, a coroutine is a function which can be paused and then later resumed. This is known as *yielding* or *suspending*. Due to ambiguity, some denote this to mean a function which has explicit syntax for pausing and resuming (this corresponds to cooperatively scheduled tasks), and others denote this to mean any function can be paused and resumed at any time, even if the pause is performed implicitly by a language runtime (this includes preemptively scheduled tasks)[19, para. 9]. For this work, the former definition will be used. On top of this, there are two types of coroutines: *stackless* and *stackfull*.

b) Stackless: Stackless coroutines only allow coroutines to yield/suspend from within a coroutine. This type has the benefit of being lightweight as very little state is retained per coroutine. The coroutine's state is commonly stored on the heap. However, due to their nature, stackless coroutines fall

prey to the function coloring problem[20] as regular functions cannot yield/suspend; only those denoted as coroutines can yield/suspend a coroutine.

c) Stackfull: Stackfull coroutines, also taxonomically known as *fibers*, allow the coroutines to yield/suspend anywhere on the stack. This is through *complete stack retention*. Every time a coroutine yields/suspends, the current call stack is retained. When the coroutine is resumed, the retained call stack is used. This adds overhead to each coroutine, as memory must now be allocated to store the state of the call stack at the moment the coroutine yields/suspends. However, this removes the function coloring problem[20] as coroutines can yield/suspend without denoting they are a coroutine.

d) Goroutines: Goroutines are a Go language feature which enables concurrency and parallelism. Similar to processes and threads, goroutines are preemptively scheduled. Despite this, goroutines are implemented as part of the language runtime (multiplexed onto OS threads) instead of as an OS primitive, also known as virtual threads or green threads in other languages. While goroutines are often considered coroutines, most akin to stackfull coroutines, goroutines, in their modern state (the goroutine scheduler has evolved over the years and as of Go 1.14, Go scheduling is non-cooperative preemption), are more in line with processes and threads[19, para. 10].

4) Event Loops: When using an event loop concurrency model, a language supports concurrency but does not support parallelism. Like coroutines, event loops are managed by a runtime and are single-threaded. Unlike all other concurrency models, event loops do not update according to a scheduler. Instead, event loops update based on an event cycle. When using an event loop, work is first published to the event loop. This work is then given to background workers, commonly OS threads, to perform the work and continue the event loop. Once the work is done and published back to the event loop, the result is given to the piece of code that was waiting for said work on the next event loop cycle. Due to this nature, many common event loop based runtimes, such as Javascript runtimes, are strictly single-threaded and disallow for parallelism.

VI. Conclusion

With the abundance of technology in the modern world, understanding programming language identity allows developers to understand the software they create. This work only scratches the surface of the many design and implementation details that subtly change the identity of a programming language by covering two foundational factors that affect a language's identity: programming paradigms that classify programming languages based on their features and features that allow semantics to convey complex ideas inside program logic. Understanding the power and limitations of the programming languages that we, as developers, use every day, will enable us to find new and innovative solutions for tomorrow's problems.

References

[1] Martin Fowler. *Language workbenches: The killer-app for domain specific languages?* June 2005. URL: https://martinfowler.com/articles/languageWorkbench.html.

[2] Robert Nystrom. *Crafting interpreters*. Genever Benning, 2021.

[3] David Walker. *Intermediate Representation*. Feb. 2003. URL: https://www.cs.princeton.edu/courses/archive/spr03/cs320/notes/IR-trans1.pdf.

[4] Mayank Bhatnagar. *Magic lies here - statically vs dynamically typed languages*. Sept. 2018. URL: https://medium.com/android-news/magic-lies-here-statically-typed-vs-dynamically-typed-languages-d151c7f95e2b.

[5] Ovid. *What to know before debating type systems*. Aug. 2010. URL: https://blogs.perl.org/users/ovid/2010/08/what-to-know-before-debating-type-systems.html.

[6] J. Roger Hindley. "The principal type-scheme of an object in combinatory logic". In: *Transactions of the American Mathematical Society* 146 (Dec. 1969), p. 29. DOI: 10.2307/1995158.

[7] Robin Milner. "A theory of type polymorphism in programming". In: *Journal of Computer and System Sciences* 17.3 (Dec. 1978), pp. 348–375. DOI: 10.1016/0022-0000(78)90014-4.

[8] Conor Hoekstra. *Codereport/plgraph*. Jan. 2023. URL: https://github.com/codereport/plgraph/tree/main.

[9] Joseph D. Darcy. *JEP 126: Lambda Expressions & Virtual Extension Methods*. Nov. 2011. URL: https://openjdk.org/jeps/126.

[10] Steve Klabnik and Carol Nichols. "The Rust Programming Language, 2nd edition". In: 2nd ed. No Starch Press, 2023.

[11] Leonhard Holz. *Well-structured Logic: A Golang OOP Tutorial*. Mar. 2020. URL: https://www.toptal.com/go/golang-oop-tutorial.

[12] Matija Pretnar. *An Introduction to Algebraic Effects and Handlers*. Tech. rep. University of Ljubljana, 2015.

[13] Gabriele Pappalardo. *Extending Kotlin with Algebraic Effect Handlers*. 2023.

[14] Daan Leijen. *Algebraic Effects for Functional Programming*. Tech. rep. MSR-TR-2016-29. Aug. 2016, p. 15. URL: https://www.microsoft.com/en-us/research/publication/algebraic-effects-for-functional-programming/.

[15] *Effekt Language*. URL: https://effekt-lang.org/.

[16] Dan Abramov. *Algebraic effects for the rest of Us*. July 2021. URL: https://overreacted.io/algebraic-effects-for-the-rest-of-us/.

[17] Herb Stutter. *The Free Lunch is over: A fundamental turn toward concurrency in software*. Mar. 2005. URL: http://www.gotw.ca/publications/concurrency-ddj.htm.

[18] URL: https://docs.python.org/3/glossary.html#term-global-interpreter-lock.

[19] without boats without. *Why async rust?* Oct. 2023. URL: https://without.boats/blog/why-async-rust/.

[20] Robert Nystrom. *What color is your function?* Feb. 2015. URL: https://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/.

Editors

Neha Kaushal

Neha is a Fulbright Visiting Doctoral Research Fellow at UC in the Department of Chemical & Environmental Engineering. Aside from her studies, she loves the outdoors and dabbles in painting and photography.

Thank you to our editors for making this magazine possible!

Phuc Mai

Phuc is a 1st year at UC studying Cybersecurity Engineering. He is involved in clubs such as MakeUC, BearcatCTF, and Cyber@UC. In his free time, he enjoys photography and playing the piano.

Akshata Upadhye

Akshata, a 2022 MS CS graduate from UC, is currently a Data Scientist at Randstad & an editorial board member for several journals. She has published research in reputable journals, spoken at numerous conferences, & received various awards for her contributions to the field.

Editorial Team

Editor-in-Chief

Natalia Lui

Natalia is a 4th year in Rhetoric and Professional Writing. Her work in professional document design and copyediting led to her involvement with the IEEE Student Magazine in 2021.
In her free time, she plays video games.

Project Manager

Mettika Ukey

Mettika is a 2nd year studying Electrical Engineering. She is involved in various organizations on campus, including IEEE, AWOCE and UC Rocketry. In her free time, she enjoys playing the piano and making art.

Contents

IEEE
University of Cincinnati Branch