EMRE SAFA ÇELİK

SOFTWARE DEVELOPER

https://github.com/EmreSafaCelik

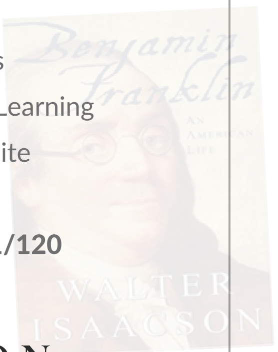
CONTACT

+90 (543) 361 3673
safaemrecelik@gmail.com
Nilüfer/BURSA

SUMMARY

I am a developer with four years of on and off startup experience where I had to take on responsibility and accountability. I believe in deep, focused work and vicarious learning, mostly through reading.

SKILLS



- React Native and Kotlin
- Next.js and React
- Node.js and Express
- Deep and Transfer Learning
- TensorFlow and TFlite
- CI/CD Essentials
- **TOEFL SCORE: 101/120**

EDUCATION

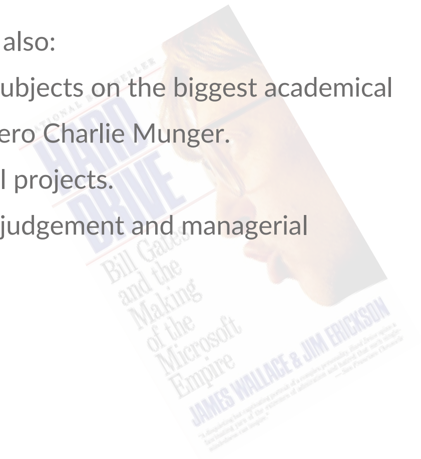
2020 - 2024
Bursa Uludağ University
BACHELORS IN COMPUTER SCIENCE

EXPERIENCE

DECIDED TO TAKE SOME TIME OFF FOR READING

Home Sweet Home

2024 June - Present, planning to stay in this position until 2025 starts

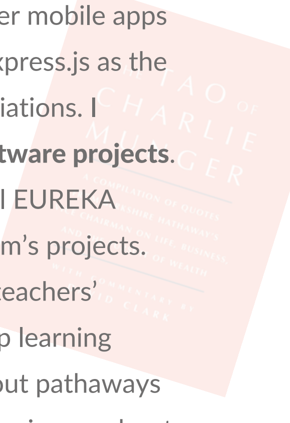


- Reading biographies, mostly. But also:
- Reading about the introductory subjects on the biggest academical disciplines, as suggested by my hero Charlie Munger.
- Allocating some time for personal projects.
- Reading about psychology of misjudgement and managerial psychology.

SOFTWARE DEVELOPER

Açık Atölye Tech

2020 - 2024 June, on and off, with more than a few sleepless nights



- I started off here by working on a mobile app for a customer. We essentially developed the whole app with three people. I was the one mostly responsible for the whole frontend, which was not insignificant. It was quite the dive into work life.
- Since then we have done many great things with the amazing people here. For example I was highly involved with the submittance of a **1507 R&D TUBITAK project**, which was later approved with a considerable budget. I was also the one to design the initial sketches of its **database design and AWS architecture**.
- We had quite a few customer projects, mostly either mobile apps or web apps. We used Next.js, React Native and Express.js as the technologies, although there were some minor deviations. I **learned many lessons on meeting deadlines of software projects**.
- I **gave seven live speeches**, one at the International EUREKA Innovation Summit on Istanbul about one of our firm's projects. Two to the general public at the firm and one at a teachers' convention about the essential math of AI and deep learning (which is a subject I just find so enjoyable), two about pathaways to learning programming, also a last one to some foreigners about various computer science concepts in English.

PERSONAL BELIEFS

You don't really need to read the rest of the CV, I will no longer talk about my qualifications, but my beliefs and thoughts.

"Curriculum Vitae" means "course of life", and I believe it also encompasses my beliefs and ideas

A FEW IDEAS ON SOFTWARE DEVELOPMENT

Don't write abstractions until you see clear signs that you need them

Abstraction is an amazing concept. It is the reason that developers can develop applications fast, users can use computers without understanding one bit what is going on under the hood. But in the world of programming, the excess use of abstractions seems to have pervaded almost everywhere. If you start a project by writing the abstractions you are guessing that you will need, then you are effectively trying to solve problems you aren't even sure exist. These abstractions later make the code much harder to understand, as one has to hop from file to file and hold so many different variables in their head. Worst of it all, when you have to make a change in an abstraction written earlier you will have to refactor all parts of the code that uses the abstraction. This situation causes such a huge loss of time, making deadlines harder to reach and costing companies a lot of money.

Err on the side of writing bad code - perfect code provides no feedback and costs time

This idea might be considered an extension to my former point. Let's assume our team is writing amazing code that never needs much refactoring, with great engineering. Well, what is actually happening is that the team is spending a lot of time, even at the not-so-important parts of the software, to make sure the code is awesome. In our industry, time means money. If a team spends a lot of time at unremarkable points of a software, then the team is effectively costing the company money, while bringing almost no benefit. Also the team is getting little to no feedback because features are seen by the shareholders/users less frequently (because of slower releases) and the only way to scale development speed is to hire more programmers, who will have to take a lot of time understanding all the abstractions before joining the team for development.

On the other hand, if a team writes code quickly then it recieves feedback quickly. Both from users and from the problems they are having with the bad code themselves. If a part of the code is always causing problems because of the quickly written code, you can then refactor the code and add abstractions. This way, time is not lost by creating abstractions in places where they are not needed.

Bugs should be solved one layer deeper

Imagine you're maintaining a critical alert system for a security control room that handles real-time threat data. You notice a recurring issue in the system logs where an alert is missing key data. A quick fix might involve adding a check to ignore such incomplete alerts: if (!alertData) return null. However, upon further investigation, you discover that the issue arises because the data-fetching process updates the state in two steps – first clearing the alert data and then attempting to fetch new data. By reversing these steps, ensuring the new data is fetched before clearing the old data, you ensure that no alert is ever processed with missing information.If you continue applying quick fixes, your system will become increasingly fragile. On the other hand, by digging deeper and addressing the root cause, you maintain a robust system with a comprehensive understanding of its critical data handling processes.

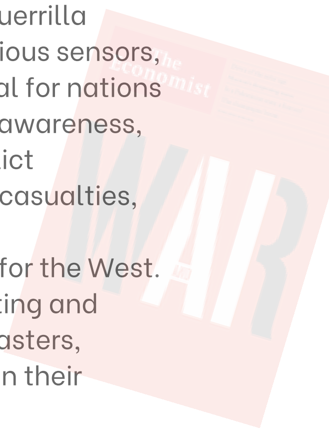
PERSONAL PROJECTS & NOTES

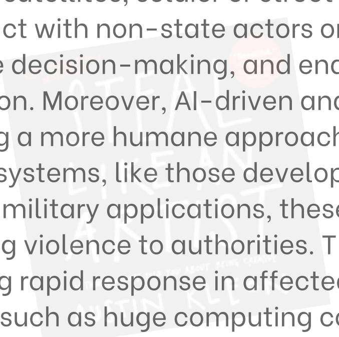
This page showcases a personal collection of general ideas that have prioritized spots in my mind. Which also means stuff I'd especially like to work on.

PERSONAL NOTES - OPPURTUNITIES FOR CIVIC IMPROVEMENT

Disaster Prevention and War

Each passing decade makes it increasingly clear that urban and hybrid warfare are on the rise, with strategic objectives now the primary determinants of war outcomes, rather than guerrilla tactics. This shift highlights the need for advanced AI systems to analyze data from various sensors, such as satellites, soldier or street cameras, and social media. These systems are crucial for nations in conflict with non-state actors or terrorist organizations, as they enhance situational awareness, improve decision-making, and enable precise targeting, leading to more efficient conflict resolution. Moreover, AI-driven analytics help minimize collateral damage and civilian casualties, ensuring a more humane approach to modern warfare.

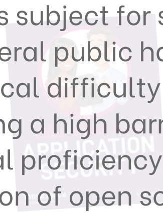




Such systems, like those developed by Anduril Inc., are already transforming warfare for the West. Beyond military applications, these systems can improve safety in civic areas by detecting and reporting violence to authorities. They can also identify fire hazards or other natural disasters, enabling rapid response in affected areas. But of course, such systems have obstacles in their course, such as huge computing cost.

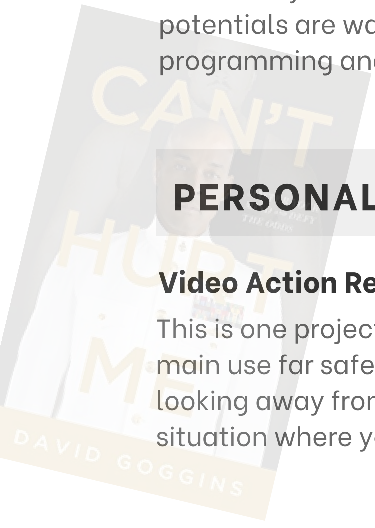
Just to give a quick baseline for such a system: The simplest approach would be to train distinct models to detect various objectives like assaultmen, logistics vehicles, etc. and develop a method to gather the data and run them one by one through each model, finding out which ones are firing high probabilities, then figuring out whar the data last fed to the model was from.

Making Open Source Tools More Accesible for General Public, Especially on AI



I have been turning the gears in my head about this subject for some time now, as I use more and more AI tools. The point here is to make sure the general public has access to these tools to be able to use them on expert level without too much technical difficulty. Right now, most open source AI tools require extensive installation processes, causing a high barrier to entry. Even after installation, the programs are hard for people with average digiral proficiency to understand.

Another obstacle in the path of widespread adoption of open source tools is the lack of marketing. Obviously there is no incentive to market such tools to the general public unless the tool doesn't have a way to monetize them, and most tools don't. Many ready-to-use open source tools' potentials are wasted because they are used by the niche people who are involved both in programming and the specific are the tool is built for.



PERSONAL PROJECTS

Video Action Recognition Based Mobile Device Control Through Camera

This is one project that I have already trained the models and developed the system for. For me, its main use far safe driving - being able to control my phone with head gestures as I drove, without looking away from the road - but it can be used for various purposes, one example being any situation where your hands or eyes are busy.