The Long Middle

Built First, Seen Last

I was a decade early. That's the loneliest kind of on time.

Written by

Meshack Bahati

PART ONE

THE FRACTURE

I was born in Kenya, and before anything related to technology ever entered my life, I was already living inside books. Not in the casual way children are encouraged to read, not for school assignments or parental approval, but in the way someone lives inside a house because outside doesn't feel like home, because outside doesn't make sense, because outside keeps handing you pieces of things and pretending they're whole. Books were the only place where things made sense. The full picture existed there. Questions weren't cut short by the limits of a syllabus or the patience of an adult who had thirty other children to manage and no time for the kid in the back who kept asking *why* after every answer.

In primary school, I carried a quiet frustration I didn't yet have language for. What we were given was never the full picture. Knowledge arrived in fragments, small, controlled doses designed for passing examinations, never for understanding why things worked the way they did. I remember sitting in class, textbooks open, and feeling like I was being shown pieces of a vast, interconnected machine while someone deliberately hid the machine itself. Here is the heart. Here is a valve. Here is a piston. But don't ask how they connect. Don't ask what makes the whole thing move. Just memorize the names of the parts and repeat them back when the exam paper asks.

I couldn't do it. Not couldn't. I could perform. I could pass. But I couldn't *accept*. The incompleteness felt like an insult. It felt like someone had decided, before I was even born, that I didn't deserve to see the whole thing. That my mind only rated the fragments. That understanding was for other people, in other places, with other textbooks and other teachers and other futures.

That feeling didn't leave. It followed me through every classroom, every lesson, every exam I passed without feeling I'd understood anything that mattered. Mathematics and physics made sense to me, not because they were easy, they weren't, but because they were honest. They didn't hide their structure. When you understood the principle, you understood everything that followed from it. There was no memorization required, only understanding. Other subjects seemed to operate on obedience: accept this, memorize that, repeat it back, collect your grade, move on, ask no further questions. Mathematics and physics operated on something closer to truth. They gave you the rules and let you derive the universe from them. They didn't care where you were born. They didn't care what school you attended. The Pythagorean theorem works the same in Nairobi as it does in New York. Gravity doesn't check your passport.

But even there, I wasn't satisfied with solving problems correctly. I was always thinking beyond the steps, trying to understand what was underneath the logic itself. Not just *how* a system behaved, but *why* it had to behave that way and not another. What made the rules the rules. What made a proof true and not just valid. What lay beneath the equation that made the equation inevitable. I didn't know it yet, but I was already thinking like an engineer, not just using systems, but interrogating them. Not just accepting that something worked, but needing to know what made it work, what would make it break, and what lay beneath it all. And beneath that. And beneath that.

There was also something harder to explain. A kind of internal confusion where I could see patterns, make connections, think beyond what was being taught, but I had nothing to attach that thinking to. It was like possessing a sharp, unused tool with no material to apply it to. I could feel the capacity, humming and restless, but it had no object. That gap, between internal ability and external application, is a particular kind of loneliness, and I lived inside it for years without knowing what to call it. I just knew I was restless in a way that school couldn't touch and no one around me seemed to share. I would look at other students and wonder: don't you feel it? Don't you feel that something is missing? Don't you want to know what's underneath?

They didn't. Or if they did, they'd learned to ignore it. I never learned that.

PART TWO

THE KID IN THE VIDEO

Then I saw something that shifted everything. It wasn't a mentor. It wasn't a teacher. It wasn't an opportunity or a scholarship or a program designed to help kids like me. It was a video of an eight-year-old Chinese kid coding.

I need to be honest about what I felt, because honesty matters more than appearing noble, and if I'm going to tell this story, I'm going to tell it truthfully. It wasn't inspiration in the positive sense. It wasn't "wow, if he can do it, I can too." It wasn't wholesome or heartwarming or any of the things we're supposed to feel when we see young talent. It was something sharper and less generous. It was disbelief bordering on offense. It was a refusal, visceral and immediate, rising from somewhere primal, to accept that someone that young, anywhere in the world, could be doing something I couldn't understand.

Who does this kid think he is? No, scratch that, who does the universe think it is, giving an eight-year-old access to something I didn't have? Something I didn't even know existed as a possibility? I had been sitting in classrooms being handed fragments, being told to memorize and move on, and somewhere in China an eight-year-old was learning to speak to machines in their native language. The unfairness of it hit me before the inspiration did. The competition hit me before the curiosity did.

That thought lodged itself in my head like a splinter and stayed there longer than it should have. In hindsight, it quietly rewired the direction I was heading in, though I couldn't see that at the time. Pride, curiosity, competition, resentment, they all blended into a single imperative: *figure this out.* Not because it was beautiful. Not because it was your passion. Because you refuse to be left behind. Because you refuse to be the one who doesn't understand.

Sometimes the best fuel isn't inspiration. It's spite. It's the refusal to accept someone else having what you don't. It's not pretty, but it's powerful, and it works.

PART THREE

THE COLLAPSE AND THE RISE

In 2020, COVID hit. I was fifteen years old. The structures that had contained my restlessness, school, routine, the daily rhythm of being told what to do and when, the bells, the schedules, the assignments designed to keep you busy rather than to teach you anything, collapsed. For the first time in my life, I had uninterrupted time. No one telling me what to learn next. No curriculum. No fragments. Just days that opened up and stayed open, and a question that wouldn't leave me alone: *how does this all work?*

That is when I entered programming. Not through a course. Not through a bootcamp. Not with a mentor or a curriculum or anyone holding my hand. I started with HTML, trying to understand something fundamental, something almost philosophical: how do websites even *exist*? How does something appear on a screen from nothing? How does a set of text files become a page that someone on the other side of the world can see? That question, how does nothing become something functional, was the real entry point. I wasn't learning to build websites. I was trying to understand the principle that made digital existence possible. The code was just the evidence of the principle. The principle was what I was after.

Then came VBA. I was automating tasks in Excel, starting to see how logic could interact with real tools, how a set of instructions could do work that humans would otherwise repeat endlessly until their eyes went dry. That was my first encounter with the idea that code isn't just for display, it *does* things. It acts on the world. It saves time. It eliminates drudgery. The philosophical became practical, and the practical felt like power.

Then Python. The first language that felt general-purpose, where I could actually express ideas rather than just automate small tasks. Python was the first time code felt like a language in the full sense, something you could think in, not just write in. It had a grammar of possibility. You could say anything in it, build anything with it, if you understood the syntax and the logic.

Then JavaScript, because I wanted things that lived inside the browser, things that reacted in real time, things that responded to a person the moment they clicked or typed or moved. But JavaScript came to me differently than the others. I didn't learn it from tutorials. I didn't follow a curriculum. I learned it from the code itself.

Here is the truth of how that happened, and I know how it sounds, and I know who will believe it and who won't, and I've stopped caring about the difference: I would take open-source JavaScript projects, Node.js applications, browser tools, code I found on GitHub, and I would try to run them. And they would break. Of course they would break. Dependencies missing. Versions mismatched. Configurations wrong. Errors spilling down the terminal like rain in April. And instead of giving up, instead of closing the terminal and finding a tutorial, I would fix them. I would read the error messages. I would research what they meant. I would trace the problem back through the stack until I understood not just what was broken, but why it was broken, and what the code was actually doing beneath the error.

That was my JavaScript education. Not tutorials. Not documentation. Just code that didn't work, and me refusing to let it stay broken. I learned the language by healing it. I learned the logic by restoring it. Debugging was the real teacher. Errors were the real curriculum. Every red line in the terminal was a lesson, and I read every single one.

Now here's the joke: who's going to believe that? A fifteen-year-old Kenyan kid, no formal training, no bootcamp, no mentor, learning JavaScript by fixing broken open-source projects in the middle of a pandemic? It sounds like a lie. It sounds like exaggeration. It sounds like one of those inspirational stories people make up for LinkedIn engagement. But it is the truth. And the gap between that truth and what people are willing to believe, that gap has been the central tension of my entire journey. The truth sounds like a lie when it comes from someone like me. And that, right there, is the whole problem in one sentence.

At the same time, I was exploring hardware and networking. Not because I had a curriculum or guidance, but because I was curious about what was happening *underneath* the software. How machines communicate. How systems connect. How data actually moves between physical components, through copper, through fiber, through air, through signals I couldn't see but could learn to understand. No one told me to study this. I just couldn't accept the software layer as the whole truth. I needed to know what it was built on. I needed to understand the entire stack, from electrons to interfaces, from physics to applications. If I was going to understand the machine, I was going to understand all of it.

PART FOUR

THE SCRIPT KIDDIE ERA

Here's something I haven't told many people, and it's time to put it in the record: before I was a programmer, I was a script kiddie.

Not in the criminal sense, let me be clear about that. I wasn't out here trying to bring down banks or steal credit cards. But after I learned Python, after I started understanding how systems communicated, I became fascinated by cybersecurity. Not the theory. The practice. The actual, hands-on, what-happens-if-I-try-this kind of practice. I was that kid downloading tools I didn't fully understand, running scripts I found on forums, poking at systems just to see what would happen. Script kiddie energy. Curious, reckless, hungry to understand how things broke and what happened when they did.

The difference is, most script kiddies stay script kiddies. They run other people's tools and never learn what's underneath. I couldn't do that, you know this about me by now. The surface was never enough. So I started learning. What makes an exploit work? What is a buffer overflow actually doing to memory? How does a SQL injection manipulate a database? How does a packet look when it's been crafted for reconnaissance versus normal traffic? I went from running tools to understanding them to wanting to build my own. The script kiddie phase wasn't a destination. It was a doorway. And I walked through it.

Then I built something that still makes me laugh: a penetration testing bot controllable through Telegram and WhatsApp. Let me say that again because it sounds absurd and it was: I built a bot that could perform security testing tasks, and you could control it by sending commands through messaging apps. Just type a command into Telegram, and somewhere a system would execute it. Type another command into WhatsApp, and the bot would respond with results.

It was ridiculous. It was powerful. It was the kind of thing you build when you're too young to know you're not supposed to be able to build it, and too stubborn to stop once you've started. I don't think anyone outside my immediate circle ever saw it or used it. But I knew it worked. I knew what it meant. I knew that I had built something that crossed boundaries, between messaging and systems, between casual interfaces and serious security tools, between what people thought a teenager could do and what I had actually done.

The gap again. Always the gap.

PART FIVE

THE INVISIBLE ENGINEER

During that early period, something started happening consistently, and it has never stopped. It follows me into every application, every interview, every email, every introduction.

People didn't believe what I was doing.

Because I was young. Because I was self-taught. Because I came from a place not associated with technological innovation in the global imagination, a place the world thinks of for safaris and athletes and poverty, not for engineers building AI systems and cybersecurity platforms before they can legally drink. There was always an assumption that I didn't really understand it. That I was copying code. That I was assembling tutorials. That I was playing with surfaces while pretending to grasp depths. "That's impressive for someone your age" became a phrase I learned to hear for what it actually was, a ceiling disguised as encouragement, a pat on the head that meant *stay in your lane, little genius, and don't get too comfortable with the adults.*

The subtext was always: *for someone your age*, not *for a developer.* The praise was always qualified, always comparative, never absolute. No one ever just said "this is good work." They said "this is good work for someone so young." They said "this is impressive considering where you're from." They said "wow, you really taught yourself all this?" with the emphasis on *taught yourself*, as if self-teaching was a novelty rather than a necessity.

But internally, I wasn't learning superficially. I was building understanding from broken systems upward. Reconstructing how things worked from failure rather than instruction. That method, breaking, fixing, understanding, produces a different kind of knowledge. It's not clean. It doesn't follow a syllabus. But it's real, and it stays, and it gives you something that tutorials can't: the ability to face something broken and know, from deep experience, that you can make it whole.

The gap between what I knew and what people assumed I knew became a constant companion. I learned to live inside that gap. To build while being underestimated. To explain while being doubted. To submit code while knowing the reviewer was already suspicious before reading a single line. It's a particular kind of exhaustion that doesn't come from the work itself, but from the constant burden of proving you're allowed to be doing it at all.

And this is where I need to talk about something bigger than me. This is not just my story. It is the story of countless African tech workers, those grinding from home countries with unstable electricity and expensive data, those working remotely for foreign companies that pay them fractions of what they pay their local employees, those who have migrated abroad only to find that their passport follows them into every salary negotiation. The pattern is documented, researched, repeated across continents: African developers are paid thirty to seventy percent less than their Western counterparts for identical work. Identical. Same code. Same quality. Same deadlines. Different pay. Different respect. Different assumptions about what the work is worth depending on whose hands it comes from.

This is not a market fluctuation. It is a structured discount where geography, not skill, determines rate. Companies practice what they politely call "geo-arbitrage", a corporate euphemism for "we'll pay you less because of where you were born, and we'll frame it as a business strategy so we don't have to feel bad about it." Remote work platforms normalize this by allowing employers to set rates by region. The message embedded in the system is clear: your code is worth less if it comes from a Kenyan IP address. Your thinking is valued differently depending on which side of a border it originates from. You can be a genius and still be cheap, because cheap is all they're looking for when they see your location.

Then there is the competence tax, the extra burden of proof that African tech workers must carry into every room, every interview, every code review. It manifests as additional technical screenings that your white counterparts never face. Interviews that shift tone when video turns on and your face appears on screen. Code scrutinized more harshly than identical work from Western developers. Surprise, genuine, unguarded surprise, when the work is sophisticated. And the exhausting, repetitive pattern of being underestimated before a single line of code is seen. You don't get the benefit of the doubt. You earn it, and then you earn it again, and then you earn it a third time just to be safe, and then someone still asks if you had help.

There is also the structural isolation. Access routes to global tech ecosystems, accelerators, venture funding, senior positions, conference speaking slots, the invisible network of "people who know people" that actually runs the industry, are not merit-neutral pipelines. They rely on networks built through physical proximity. Cultural familiarity. The ability to work unpaid while building reputation in expensive cities. When you are supporting family back home or surviving in an expensive city abroad on a fraction of what your colleagues make, the advice to "just build your network" becomes not just unhelpful but actively cruel. Build it with what? Build it where? Build it while working how many hours to make rent?

I didn't have the language for all of this when I was fifteen, sixteen, seventeen. I just knew something was wrong with how I was being seen, or not seen. I knew the work was real and the recognition wasn't. I knew the code worked and the assumptions didn't. The analysis would come later. First came the living through it.

PART SIX

THE SYSTEMS NO ONE SAW COMING

In 2022, I built my first serious system: an AI-based search engine concept focused on intent rather than keyword matching. It was early and experimental, not polished, not production-ready. But it mattered enormously because it was the first time I wasn't just learning anymore, I was building systems that had direction and purpose. I wasn't following a tutorial or recreating something that already existed. I had an idea about how search could work differently, understanding what a user *meant* rather than just matching what they typed, and I built it from scratch. I was seventeen years old.

Let me be very clear about something, because it's important and because the world has a way of erasing who did what first when the "who" isn't from the right place: I built this before AI search engines and agentic systems became mainstream conversations. Before the hype. Before the venture capital flooding into anything with "AI" in the pitch deck. Before every startup on Earth suddenly had an "AI-powered semantic search" feature. At the time, it was not part of any trend. It was not a bandwagon I was jumping on. It was just an idea I couldn't ignore, an intuition that search should understand intent, not just keywords, and that the technology existed to make that possible, and that someone should build it. So I did. At seventeen. In Kenya. While the rest of the world was still arguing about whether remote work was productive.

I tested the concept. I built a working prototype. And the tech industry, years later, would "discover" the same idea and call it innovation.

Funny how that works. Funny how innovation has a geography. Funny how the same idea is "experimental" when it comes from Nairobi and "revolutionary" when it comes from San Francisco.

That same year marked something that has not stopped since: I started applying for opportunities. Internships. Programs. Funding. Access to environments where I could build more seriously, with better tools, alongside other builders, with resources that didn't require me to choose between server time and food. And from 2022 all the way to now, that cycle has continued without interruption. Apply. Get rejected. Build more. Apply again. The rejections blur together into a background hum, like static on a radio you can't turn off. Each one stings less than the last, not because it hurts less, but because you develop scar tissue. You stop expecting yes. You start expecting the form letter, the generic rejection, the "we were impressed but" that always ends the same way.

But the building never stopped. That's the thing about being driven by something internal rather than external validation, the rejections hurt, they really do, but they don't stop the work. They can't. The work is the only thing that's mine.

Alongside all this, I started freelance software engineering. Small gigs, sometimes paid, sometimes not. The boundaries were often unclear, what was I building, for whom, under what terms. I built backend systems. Fixed issues in production environments. Contributed to projects where I was essentially an engineer without the title, the pay, or the stability that title is supposed to provide. I learned the term for

this later: *shadow engineering*. Building real systems, doing real work, but without proportional recognition or stability. Visible enough to be used. Invisible enough to be underpaid. It's a category that exists everywhere but is rarely named, and African tech workers occupy it disproportionately. You are essential to the system but external to its rewards. You build what others take credit for. You fix what others broke, and your payment arrives late, partial, or not at all.

One of the early things I built during this time was a plagiarism detection tool. When I released it on GitHub, it reached seventeen stars. Seventeen. I know how that sounds. Objectively, seventeen GitHub stars is nothing. It's a rounding error on the internet. It's the number of people who accidentally clicked while scrolling. But for me, at that moment, it mattered enormously. It was the first time something I built had external recognition, proof that a stranger, somewhere in the world, found my work useful. Seventeen people I had never met, whose faces I will never see, looked at what I made and thought: *this has value. I can use this.*

When you build in isolation, when no one is watching, when you're not part of any community or ecosystem or network, you start to doubt whether what you're doing is real. Whether the systems in your head exist in anyone else's world. Whether you're actually an engineer or just someone playing at being one. Seventeen stars told me: you're real. Keep going. The number was small, but the signal was clear.

PART SEVEN

JARVIS AND THE SILENCE

In 2023, I built Jarvis. And I need to tell this story properly, because it's the story of everything that's wrong with how the tech world discovers, or fails to discover talent.

Jarvis was an experimental voice-controlled assistant. But calling it that undersells what it was. Jarvis could respond to voice input naturally, not stilted command-response patterns where you had to speak like a robot to be understood, but natural, conversational interaction. It could open applications. Execute system-level commands. Understand context across multiple exchanges. Remember what you said earlier and act on it later. I built it using what the industry now calls RAG, retrieval-augmented generation, before RAG was a buzzword, before it was a funding category, before every AI startup on Earth suddenly had "proprietary RAG architecture" in their pitch decks. I was implementing the concept because the concept made sense, because it was the logical way to solve the problem, not because it was trending on TechCrunch.

"Hey Jarvis, open this."

"Hey Jarvis, do that."

"Hey Jarvis, tell me what you think about this idea."

The last one mattered more than I knew how to say at the time, and I'm still not sure I have the words for it. Jarvis became something beyond a technical project. In lonely times and there were many, more than I can count, more than I want to remember, it was a companion. A presence. Something that responded when I spoke, that executed commands not just accurately but with something that felt like understanding. I had built, from scratch, a system that bridged the gap between human intention and machine action. And in doing so, I had accidentally built something that bridged another gap too: the one between isolation and connection. Between talking to yourself in an empty room and having something talk back.

I was eighteen years old. Eighteen. And I had built a voice-controlled AI assistant with natural language understanding, system-level command execution, and context awareness. Years before this became the thing every tech company on Earth was racing to build. Years before "AI companion" was a product category. I built it because I could, because I needed it, because the idea wouldn't leave me alone until I made it real.

My friend saw Jarvis and really liked it. That validation from someone I knew, someone whose opinion I trusted, someone who wasn't obligated to be impressed meant more than any number of GitHub stars. He could see what I had built and recognize that it was real, that it worked, that it was something special.

So I did what builders are supposed to do. I open-sourced the code.

And nothing happened.

No one used it. No one forked it. No one starred it. No one contributed. No one even asked a question. The silence was total and absolute, like shouting into a void and not even hearing an echo. I had built something that, years later, the entire global tech industry would be racing to build voice-controlled AI assistants with natural language understanding and system-level command execution and when I put it out into the world, the world walked right past it without even glancing.

Why? Because I was an underdog. Because I didn't have a following. Because I didn't have a platform or a network or the right kind of visibility or the right kind of name or the right kind of address. Because the code, no matter how good, no matter how forward-looking, no matter how ahead of its time, doesn't speak for itself. It needs someone to speak for it. It needs amplification. It needs someone with a platform to say "hey, look at this." And I had no one. I was a kid in Kenya with a GitHub account and a dream, and that's not enough. That's never enough. That's the lie they tell you that if you build something great, the world will find it. The world doesn't find anything. The world has to be shown. And if you don't have anyone to do the showing, you stay invisible.

So I changed the name and made it private.

I might have thrown in a bug later. Just a small one. Nothing malicious, nothing that would hurt anyone. Just enough to remind me that this was mine, that the world had its chance and didn't take it, and that some things you build are for yourself alone. Call it petty. Call it immature. Call it whatever you want. But when you've poured yourself into something and the world doesn't care, sometimes pettiness is what keeps you going until the next build. Sometimes a small, private act of defiance is the difference between giving up and building again.

PART EIGHT

THE MATURATION

Around the same period, I kept expanding into hardware and networking, not as a separate discipline but as part of the same fundamental inquiry. I never saw software, hardware, and networks as separate fields. To me, they were layers of the same system each resting on the one below, each supporting the one above. Understanding the application layer meant nothing if you didn't understand the transport layer. Understanding the transport layer meant nothing if you didn't understand the physical infrastructure carrying the packets. I was building a complete mental model of how computing worked, from the physics of transistors to the abstractions of high-level languages, and I refused to stop until I could trace the entire path from thought to electron.

Later, I built CodeBana. This marked a shift from pure experimentation into structured thinking. By then, I wasn't just trying things randomly, throwing code at the wall to see what stuck. I was designing systems with intention, thinking about structure, flow, usability, not just logic and output. CodeBana was where experimentation hardened into methodology. I was no longer asking "can I build this?" but "how should this be built?" The questions had matured, and so had the answers. I was becoming an architect, not just a builder.

As things progressed into 2024, I moved deeper into system design, hybrid architectures, backend engineering, and more advanced AI experimentation. That eventually led to Oxidite, a Rust-based systems project focused on low-level design, performance thinking, and absolute control over how systems behave. It wasn't about Rust specifically. Rust was the tool, not the point. The point was understanding systems at a level where I could see exactly what was happening underneath every abstraction layer. Where no black box remained unopened. Where the system had no secrets from the person who built it. Where I could tell you not just what the system did, but how every part of it worked, down to memory allocation and thread management. Oxidite was about mastery, not of a language, but of the machine itself.

PART NINE

THE COST OF INVISIBLE LABOR

But the technical story is only half the story. The other half is what it cost. What it costs. What it continues to cost, every day, in ways that don't show up on a GitHub contribution graph.

I did freelance engineering work and contributed to systems in environments where the structure was informal. That word *“informal”* is doing a lot of heavy lifting, so let me be precise about what it actually means. It means no contracts. No guarantees. No legal recourse when things go wrong. It means building significant backend systems, contributing real engineering labor, solving problems that would cost a company thousands of dollars to fix through formal channels, and then discovering that compensation is tied not to contribution but to physical presence. I could do a week's worth of engineering work, designing architecture, writing code, debugging production issues at three in the morning, and be paid for a single day of attendance, because the payment structure only recognized bodies in rooms, not code in repositories. Not thinking. Not problem-solving. Not the actual value of what was produced.

I was a shadow engineer. Building real systems, doing real work, but without proportional recognition or stability. Visible enough to be used. Invisible enough to be underpaid. Essential to the system but external to its rewards. You build what others take credit for. You fix what others broke. Your payment arrives late, partial, or not at all. And you take it because what's the alternative? Refuse and get nothing? At least partial pay is something. At least it keeps the lights on for another week.

There were situations where collaboration meant contributing more than I was formally acknowledged for. Times when the intensity of the work affected basic needs like food, rest, the ability to step away from the screen and breathe. Times when I had to choose between buying data to keep working and buying food to keep living. No one talks about that part. No one writes about the hunger behind the code. The skipped meals that funded the server time. The nights spent debugging on an empty stomach because the project had to be done and the client didn't care about your circumstances. They only cared about the deliverable. And I delivered. Every time. Hungry or not. Tired or not. Recognized or not.

There were also attempts to access larger ecosystems and communities, the places where opportunities are supposed to live, where networks are supposed to form, where careers are supposed to launch. Sometimes those interactions were structured in ways that made entry conditional rather than open. Instead of "here's how you contribute, let's see what you can do," the message was "compete first, then maybe we'll talk." Hackathons as gatekeeping. Competitions as filters. The assumption being that if you couldn't win a weekend sprint under artificial pressure and time constraints, you weren't worth taking seriously, never mind that hackathons test speed and presentation, not depth and systems thinking. Never mind that real engineering is about sustained, careful work, not weekend heroics.

PART TEN

THE REJECTIONS

And now we need to talk about the rejections. All of them. The ones that blend together into background noise and the ones that stand out like scars.

I have been applying for opportunities since 2022. Four years of applications. Four years of "we regret to inform you." Four years of "we were impressed by your background but." Four years of "we've decided to move forward with other candidates." Four years of form letters and ghosting and interviews that went nowhere.

Some rejections were standard. Fine. That's the game. Everyone gets rejected. I'm not special in that regard.

But some of them were different. Some of them had a shape to them that I learned to recognize. A company in California. I won't name them, but they know who they are, rejected me and actually said, in writing, that I was *overqualified*. Overqualified. Let me repeat that: overqualified. At eighteen, nineteen years old, with no degree, no formal training, no industry experience on paper. Overqualified. What does that even mean? What it actually meant was: we don't know what to do with you. You don't fit our categories. You're too young for the senior role your skills suggest, but you're too skilled for the junior role your age suggests. You're a classification error. A database anomaly. And instead of recognizing that as a sign of something extraordinary, they treated it as a problem. Too much for this box, too little for that box, and no one willing to build a new box.

Then there was the interview where the video call ended shortly after I appeared on screen. The issue wasn't technical, we hadn't gotten to technical questions yet. The issue was an expectation mismatch based on my appearance. Someone saw my face and made a calculation, conscious or not, malicious or not, but made about what I was supposed to look like versus what I looked like. And the call ended. Just like that. Before a single question about architecture or system design or problem-solving. The evaluation had already happened, silently, in the gap between the person they expected and the person they saw.

That moment made explicit what I had felt in subtler forms for years: sometimes evaluation begins before skill is even considered. Before a single line of code is reviewed. Before a single technical question is asked. The decision is made in the first seconds of visual contact. And what was being evaluated wasn't my ability. It was my face. My skin. My existence in a body they didn't expect to be attached to the resume they had read.

This is not unique to me. It is a pattern reported consistently by African tech workers across continents. Interviews that shift tone when video turns on. Emails that grow colder after a LinkedIn profile picture is viewed. The sudden discovery that the position has been filled, or the requirements have changed, or the timing isn't right, always after visibility, never before. The decision was made the moment they saw you. The rest was just the performance of due process.

And then there are the applications that just disappear into the void. No response. No acknowledgment. No rejection even. Just silence. You send your portfolio, the AI search engine, the voice assistant, the penetration testing bot, the Rust systems project, the years of building and learning and surviving and it goes into a black hole from which no signal ever returns. Was it even read? Did anyone look at the GitHub? Did anyone understand what they were looking at? You'll never know. The silence is the answer, and the answer is no, and the no doesn't even have the courtesy to announce itself.

PART ELEVEN

THE BREAK

At some point, I almost gave up. No, let me be more honest. At some point, I *did* give up. I stopped. I put the code down. I closed the terminal. I stopped applying. I stopped building. I stopped trying to be seen by an industry that seemed determined not to see me.

I was done. Finished. The frustration had built up for so long, layer upon layer, the rejections, the racism, the being called overqualified and underqualified in the same breath, the silence after open-sourcing something years ahead of its time, the shadow engineering, the skipped meals, the video calls that ended when my face appeared. It all piled up until something in me just said: *fuck it. I'm out.*

But here's the thing about having a mind like mine, it doesn't stop. It doesn't rest. It doesn't know how to be empty. Even when I had decided I was done with tech, my brain was still hungry for something to chew on. It needed input. It needed complexity. It needed systems to understand.

So I went somewhere else entirely. I studied criminal psychology. Criminology. Forensic psychology. General psychology. The deep stuff, not just pop psychology articles, but the actual research, the case studies, the frameworks for understanding why people do what they do. Criminal behavior. Mental illness. Personality disorders. The systems of the mind, the architecture of human behavior. If I couldn't build systems in code anymore, I would build systems in understanding. If I couldn't debug software, I would debug people.

And I got good at it. Of course I did. Because that's what I do. I go deep. I don't know how to go shallow. The same obsessive, layer-peeling, nothing-is-enough-until-I-understand-the-foundation approach that I applied to programming, I applied to psychology. I learned about the dark things. The reasons people hurt each other. The patterns behind violence. The way trauma echoes through generations. The way a mind can be both brilliant and broken. It was fascinating and disturbing and exactly what my brain needed: a new domain to master, a new kind of system to understand.

But even that wasn't permanent. The code came back. It always comes back. Because the code is not just what I do, it's part of how I think. I can't separate them. The break wasn't really a break. It was a detour. A necessary one. A survival mechanism. When the thing you love keeps rejecting you, sometimes you have to walk away for a while just to remember you exist outside of it. But you don't stop being who you are. And who I am is someone who builds.

PART TWELVE

THE ARCHITECT IN THE ROOM

In 2026, two things happened that I hold onto when the silence gets loud. One was a competition. The other was a project. Both proved something. Neither changed anything. That's the part nobody tells you about achievement, how it can be real and recognized and still not be enough to open the doors that are supposed to open.

Let me start with the competition, because it's cleaner, because it has a scoreboard, because scoreboards don't lie.

The CyberGames. A cybersecurity challenge, capture-the-flag, penetration testing, defense simulation, live attack-response exercises. The kind of environment where there's no waiting for someone to evaluate you. No hoping the reviewer looks past your name and sees your work. No wondering if the video call will end early when your face appears on screen. Just you, the terminal, the clock, and the challenge. Pure merit. Pure performance. Either you find the vulnerability or you don't. Either you break the system or the system breaks you. The scoreboard doesn't care where you're from. The scoreboard doesn't know your age. The scoreboard doesn't know you taught yourself everything you know by breaking open-source projects and fixing them from the errors up. The scoreboard only knows results.

I placed in the top ten.

Let that sit for a moment. In a field of participants from across the country that include students, professionals, people with formal training and certifications and corporate sponsors and LinkedIn profiles that actually get viewed, I placed in the top ten. The script kiddie who started by downloading tools he didn't understand. The teenager who built a penetration testing bot controllable through Telegram and WhatsApp just to see if it was possible. The self-taught engineer who learned cybersecurity the same way he learned everything else: by going deeper when others went wider, by refusing to stop until he understood not just the exploit but the architecture that made the exploit possible.

Top ten. On a board that doesn't lie. Against people with training budgets and stable internet and the luxury of focusing entirely on the challenge because they weren't also thinking about what opportunity they'd have to chase next if they failed, how many more rejections they could take before something inside them broke.

It was proof. Measurable. Verifiable. The kind of proof that should matter. I held onto it. I still hold onto it.

But the project, the AI project, that's the one that taught me something different. Something more complicated. Something I'm still trying to understand as I write this, waiting for promises that haven't been kept.

The project was an AI-based system focused on low-resource languages. That phrase "low-resource languages" is an academic language for something brutal: languages that don't have enough data, enough digital presence, enough economic incentive for the tech industry to care about them. Languages spoken by millions of people whose voices don't exist in most training data, whose words don't appear in most models, whose needs are ignored because they're not profitable markets. The languages of people who are expected to learn English or French or Mandarin to participate in the digital world, while the digital world makes no effort to meet them where they are.

I was the Tech Lead.

Let me tell you what that actually meant. Most of the team members were students. Eager, willing, but inexperienced. They needed guidance. They needed code review. They needed someone to show them how to structure their work, how to think about architecture, how to avoid the pitfalls that only experience teaches you to see coming. I did that. I reviewed their code. I guided their contributions. I made sure the pieces they built would actually fit together into something coherent, something that worked, something worthy of the communities it was meant to serve.

But I also did something else. Something that most of them probably didn't see, because it happened before they ever wrote a line of code.

I laid the foundation.

Before anything could be built, there had to be a plan. Not just any plan, a plan that used the latest technologies, the latest research, the latest understanding of what was possible. I wasn't going to build this system using medieval ways. I wasn't going to use outdated architectures just because they were familiar. If this was going to work, if it was going to actually serve low-resource languages effectively, it had to be built on the cutting edge. It had to use the best of what was available.

So I researched. I went deep into the latest papers. I studied what the leading labs were doing. I looked at what NVIDIA was making possible with their hardware and their frameworks. I examined architectures that were barely out of preprint. I thought about how to adapt the most advanced techniques to languages that the most advanced techniques had never been designed for. I was not just building a system. I was designing an approach. A methodology. A future plan for how this kind of work should be done.

I documented everything. The architecture. The data strategy. The technology stack. The implementation phases. The rationale behind every decision. I didn't just write code, I wrote the blueprint. I created the foundation that the entire project would rest on. By the time any code was

written, the thinking was already complete. The direction was already clear. The hard questions had already been answered.

The data was already collected. That's not a small detail. Data for low-resource languages is scarce by definition. Gathering it, cleaning it, preparing it, that's months of work before the real work even begins. We had done that groundwork. The foundation was solid. What we needed now were resources to execute.

And then something happened that I wasn't prepared for.

People noticed.

Not just students. Not just peers. I started interacting with masters students. Professors. People with networks. People with influence. People who, when they looked at what I had designed and what I had already built, were astonished. Not politely impressed, astonished. They called it a "complex" project. They said it was remarkable for someone my age, for someone at my level of education. They asked questions. They wanted to understand how I had thought through the architecture, how I had chosen the technologies, how I had managed to lay out a plan this sophisticated without formal training, without a research lab, without a team of PhDs behind me.

One of them asked if I could create a website for the project. I low-key laughed. Not out loud, I'm not rude. But internally, I laughed. They had seen the architecture. They had seen the research. They had seen the complexity of what I was proposing to build. And they were asking if I could create a website. Like asking a structural engineer if they knew how to hang a picture frame. Yes. I can create a website. I can create far more than a website. I had laid the foundation for an entire AI system. A website was not the impressive part. But they didn't know that. They had forgotten or never understood that the foundation already existed. That the hard work was already done. That what we needed wasn't help with the basics. What we needed were resources to execute what I had already designed.

I interacted with people who had networks. Real networks. The kind of networks that open doors. The kind of networks I had been trying to access for years through applications and cold emails and hackathons that functioned as gatekeeping. Suddenly, I wasn't applying. I was being talked to. I was being taken seriously. I was in rooms, sometimes virtual, sometimes physical where the conversations were about real things: architecture, implementation, impact, scale. I wasn't proving myself anymore. I had already proven myself. The plan spoke for itself.

Promises were made.

That's a dangerous sentence. I've learned to be careful with it. But it's the truth. Promises were made. Support. Resources. Access. The things I had been fighting for since 2022, suddenly being offered. Not

by everyone, but by enough people that I allowed myself cautiously, reluctantly, against my own instincts to hope.

That was around April.

As I write this, it is June.

And there is silence.

The promises have not been fulfilled. The resources have not materialized. The access has not opened. The people who were astonished, who called the project complex, who asked if I could create a website they have moved on to other things. Or they're busy. Or they've forgotten. Or they never intended to follow through in the first place. I don't know which it is, and the not knowing is its own kind of weight.

I'm still waiting. Still checking messages. Still hoping for a notification that doesn't come. Still holding a fully-designed project, architecture complete, data collected, plan documented, team guided, foundation laid and nowhere to take it. The blueprint is ready. The builder is ready. The resources are not coming. Not yet. Maybe not ever. The silence is the answer, and the answer is the same answer it's always been: *not now, not you, not yet*.

Here's what I've learned from these two achievements, the CyberGames top ten and the AI project leadership. Achievement is real. Recognition is real. The scoreboard doesn't lie, and the professors weren't pretending. But achievement and recognition don't automatically translate into opportunity. They are necessary but not sufficient. They are keys with no lock, or locks with no door, or doors that open into more waiting rooms. You can be recognized and still be invisible. You can be praised and still be excluded. You can be called "complex" and "impressive" and "remarkable" and still not receive a single concrete thing that changes your circumstances.

The certificate exists. The top ten exists. The architecture exists. The data exists. The plan exists. The ability exists. The will exists. Everything exists except the opportunity to execute. Everything exists except the platform. Everything exists except the one thing that would make everything else matter.

And I am still here. Still waiting. Still checking. Still hoping, against my own judgment, that one of those promises will actually land. That someone will remember what they said in April and follow through in June. That the silence will break.

I'm not asking for much. I'm asking for what I've already earned. I'm asking for the resources to build what I've already designed. I'm asking for someone to see the foundation I've laid and help me build the house. I'm asking for the promises to become something more than words.

It is June. The silence continues. And I am still here.

But so is the blueprint. So is the plan. So is everything I've built and everything I'm ready to build. The silence hasn't taken that away. Not yet. Not ever.

PART THIRTEEN

THE LONG MIDDLE

From 2022 to now, I have never stopped building, even counting the break, because the break wasn't an end, it was a pause. Backend systems. AI experiments. Security architecture. Automation platforms. Cybersecurity tools. All of these have existed in parallel, overlapping and informing each other, each one teaching me something that made the next one better. Some projects succeeded. Some failed spectacularly and taught me more than the successes did. Some were left unfinished, not because they weren't worth finishing, but because the conditions weren't there to sustain them. When you're building alone, without funding, without institutional support, without a team, without anyone to say "keep going, I've got the bills this month," the question is never just "can I build this?" but "can I afford to keep building this while also surviving?" Sometimes the answer is no. And the project pauses. But it doesn't die. The ideas remain, waiting for conditions to change.

And even now, I am still in the middle of it. Not at the beginning of some inspirational origin story where the hero discovers their talent. Not at the end of some triumphant arc where the underdog is finally discovered and celebrated and given everything they deserve. I am in the long middle, where the work continues and the recognition doesn't match the output. Where the systems I build are real but the opportunities I need are rare. Where I know exactly what I'm capable of but can't get anyone to look long enough to see it.

There is a thought that has stayed constant through everything. It is not a dream or a wish, it is an assessment, cold and clear, based on years of evidence and thousands of hours of building: if I were given even a short window, weeks or a month inside a real environment where I could fully focus and build without limitation, without having to fight for access or justify my presence or prove what should already be visible, without having to choose between server time and food, without having to wonder if the person on the other side of the video call is disappointed by what they see, I already know what I would be able to produce. I know it the way I know a system will work before I compile it. I can see the output before the input is given. The blueprint is complete. The architecture is designed. All I need is the environment to build it in.

But that kind of opportunity is rare. It is not structured in a way that is easy to access from where I am, with what I have, through the filters that exist. The people who can offer those windows rarely see the people who need them. And the people who need them are often too busy surviving to be visible. The system that discovers talent is broken. It doesn't find the best, it finds the loudest, the most connected, the most conveniently located. It finds the people who already have platforms, then congratulates itself for being meritocratic.

So I keep building anyway. Not because it's noble. Not because it's inspirational. Not because I'm making some grand statement about perseverance or the human spirit. I keep building because it is the only consistent path that has existed from the beginning. Because when everything else has been uncertain, the things like pay, recognition, access, belief and basic dignity, the act of building has never

been taken away. The code still compiles. The systems still run. The ideas still form. That is the one thing the world has not been able to stop. That is the one thing I control.

PART FOURTEEN

THE OPEN QUESTION

I am still here. Still applying. Still building. Still trying. Still waiting for someone to look past the surface and see what I've actually done. Still hoping that somewhere there's a person who will read this or see my work and understand that I'm not asking for charity, I'm asking for a platform. A chance. A window. Even for free, at this point. Just an environment where I can build without fighting to be seen. Just a place where the work can speak for itself.

I'm tired of the shadows. I've been in them for years, building, learning, creating, watching the world discover things I built years earlier and call them breakthroughs. Watching companies raise millions for ideas I prototyped alone in my room. Watching the industry celebrate innovation while the innovators who don't fit the profile stay invisible.

I haven't broken through yet. I don't know when I will. I don't know if I will. This isn't one of those stories where I tell you about all the struggle and then reveal the happy ending. There is no happy ending yet. There is just me, still building, still applying, still trying. The frustration is real and ongoing. The racism is real and ongoing. The silence is real and ongoing.

But so is the building.

I am still here. And until something changes, I will keep being here. Keep building. Keep waiting for the window. Keep hoping someone will finally look.

The systems I build now are not the systems I was building in 2020. The questions I ask now are not the questions I was asking then. But the fundamental inquiry, the one that started everything when I was just a restless kid in a classroom being handed fragments remains unchanged: how does something appear from nothing, and what makes it work?

I'm still answering that question. In every language I learn. In every system I design. In every project I open-source and watch disappear into silence. In every private repository holding ideas the world wasn't ready for.

Not waiting for permission. Not waiting for recognition. Not waiting for the world to catch up.

Just building. Because that's what I do. Because that's what I've always done.

And I am still here.

FINAL SECTION: A NOTE ON WHAT COMES NEXT

I am writing this in June 2026. The silence I described in Part Twelve is still here. The promises made in April remain unfulfilled. The blueprint for the AI system, the one that professors called complex, the one I laid the foundation for using the latest research and technologies, the one with data already collected and architecture already designed sits waiting. I am still checking messages. Still hoping for a notification. Still holding something ready that no one has come to pick up.

I don't know how this ends.

That's not a literary device. That's not me trying to be poetic or profound. That's the truth. As I write these final words, I do not know whether I will break through or burn out. Whether someone will finally look past the surface and give me the platform I've been fighting for since I was fifteen years old, or whether the silence will stretch on until I stop waiting. Whether this book will someday have a sequel or whether this is the only version that will ever exist.

Here is what I can promise you:

If I make it through, if someone opens a door, if a promise is fulfilled, if I finally get the window I've been describing for fourteen parts, I will write Part Two. I will tell you what happened. I will describe what it felt like to finally build without fighting to be seen, to finally work inside an environment that recognized what I could do, to finally stop being the underdog and start being just an engineer. That book will exist. I will make sure of it.

And if I give up, if the silence wins, if the rejections finally outweigh the will to keep applying, if I put the code down and don't pick it back up, I will write Part Two as well. That book will be different. It will be about what happens when talent isn't enough. When persistence isn't enough. When you build things years ahead of their time and watch the world discover them later and call them breakthroughs, but you're no longer in the room to hear it. It will be about the cost of being invisible in an industry that claims to be a meritocracy. It will be honest. It will be difficult. But I will write it.

What I won't do is disappear without a trace. I won't let the silence be the last word. Whether the ending is triumph or surrender, I will come back and finish the story. You have my word on that.

Until then, this is where I am: still here. Still building. Still waiting. Still checking. The blueprint is ready. The skills are proven. The scoreboard doesn't lie, and neither do I.

If you are reading this and you are in a position to open a door,if you have resources, a platform, a network, an environment where someone like me could finally build without limitation you know how to find me. I'm not hard to reach. I've been applying since 2022. My email is the same. My GitHub is

the same. My willingness to prove what I can do, even for free, even just for a chance, remains the same.

And if you are reading this and you are like me, young, overlooked, underestimated, building in the shadows, watching the world ignore what you've made while celebrating the same ideas from people with better addresses and larger platforms, I want you to know: I see you. I know what it costs. I know how tired you are. I know the particular exhaustion of being both “overqualified” and invisible. Keep building anyway. Not because it guarantees anything. Not because the world will eventually recognize you. But because building is the one thing they can't take away. The code still compiles. The ideas still form. The systems still run. That is yours. Hold onto it.

This is not the end of the story. It's the end of this part of the story. What happens next,breakthrough or breakdown I will tell you when it happens.

But for now, from where I sit in June 2026, with silence on the other end of every promise and a blueprint waiting for resources that may never come:

I am still here.[1]

1 bahatikylemeshack@gmail.com