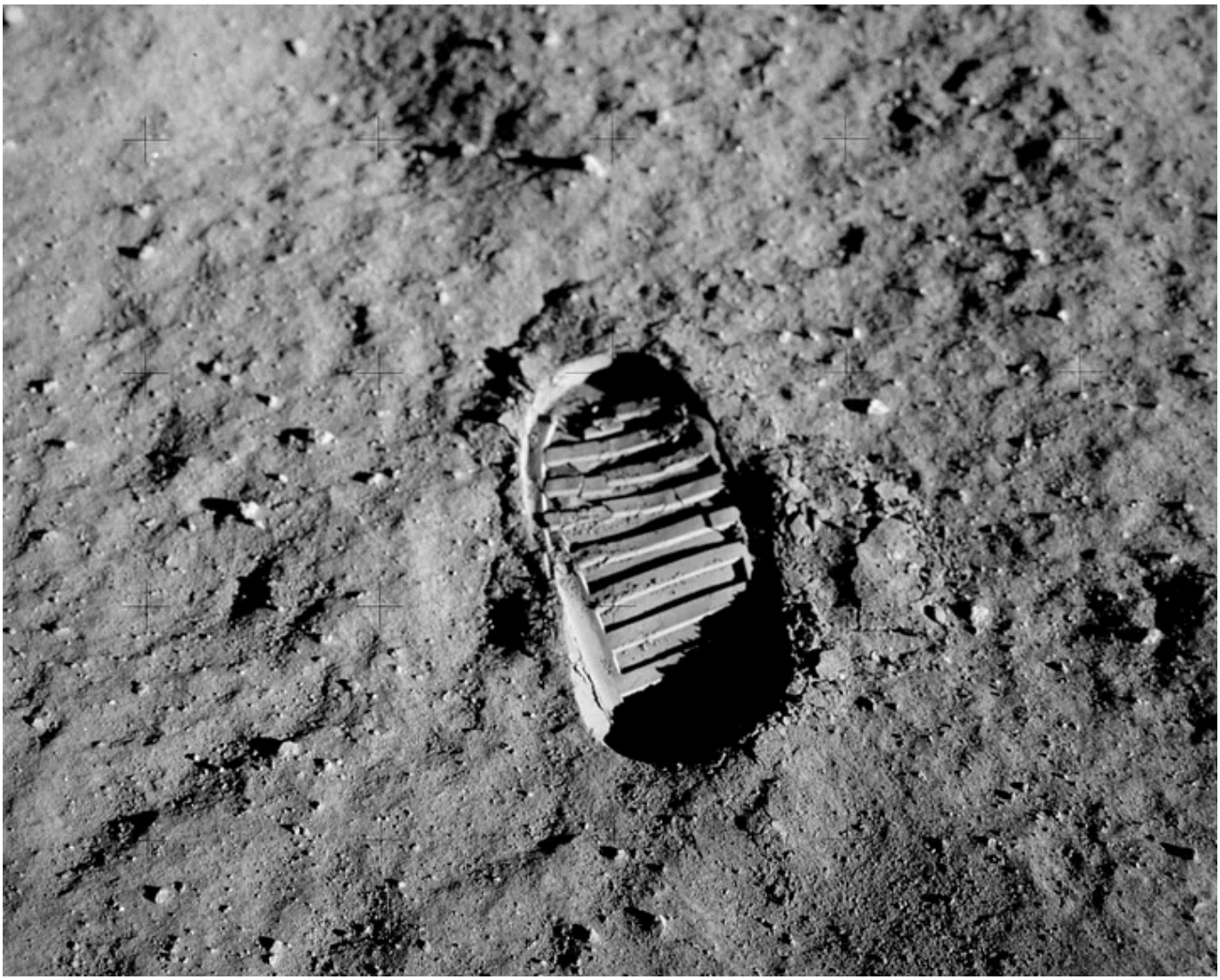
The Deep Hub

Member-only story

From Curiosity to the Moon: What I Learned Reading the Code That Landed Apollo 11

Palash Mishra · Follow

Published in The Deep Hub · 6 min read · Apr 6, 2025



A few nights ago, I found myself falling down a rabbit hole on GitHub.

I had gone there looking for something entirely different, some machine learning model to clone, maybe a React animation library. You know, the usual. But a trending repo with a nostalgic title stopped me mid-scroll:

> **Apollo-11 / The original Apollo 11 Guidance Computer (AGC) source code.**

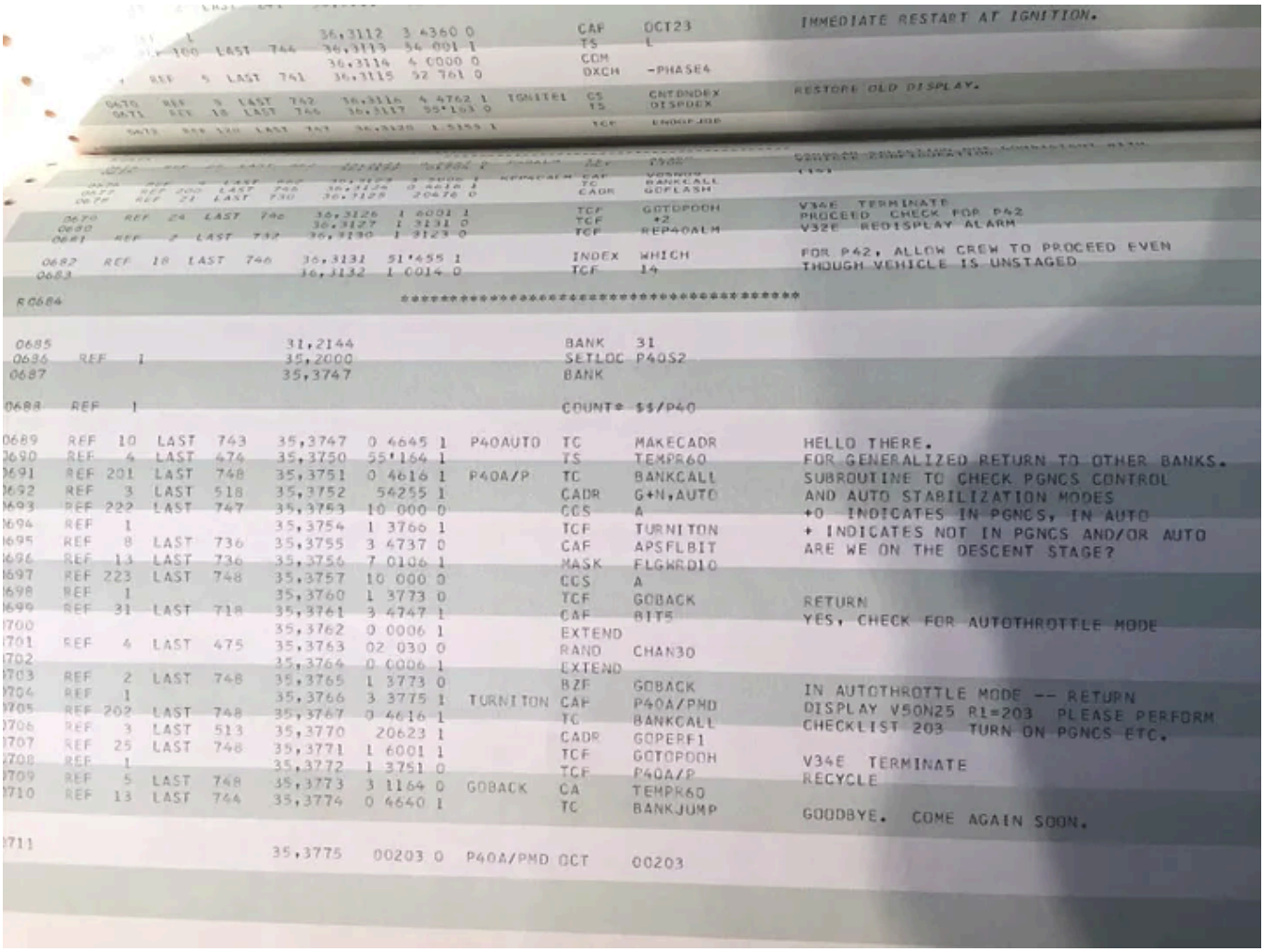
Wait, what? What I found wasn't just a codebase. It was a time capsule. And what started as a 5-minute curiosity turned into a full evening… then a night… then this blog.

A Museum in Markdown

Margaret Hamilton Led the NASA Software Team That Landed Astronauts on the Moon

I clicked and went through different modules.

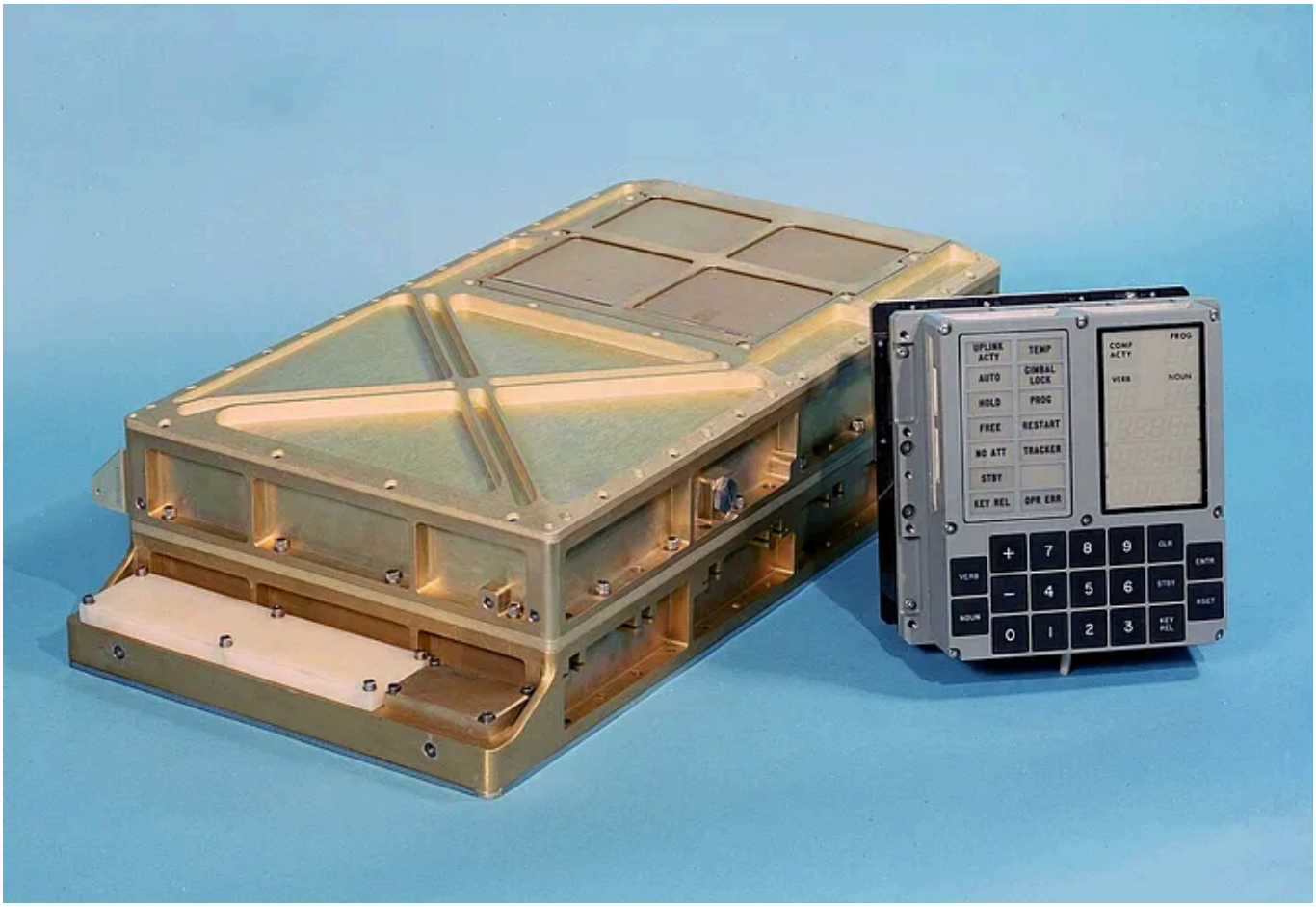
At first glance, it didn't look special. Just some assembly code in files named things like `LANDING`, `ENTRY`, `GUIDANCE`.



```
                                                  CAF     OCT23          IMMEDIATE RESTART AT IGNITION.
                                                  TS      L
                                                  COM
                                                  DXCH    -PHASE4
                                                  CS      CNTRINDEX      RESTORE OLD DISPLAY.
                                                  TS      DISPDEX
                                                  TCF     ENDOFJOB

                                                  TC      BANKCALL
                                                  CADR    GOFLASH

0670    REF  24  LAST  746   36,3126  1 6001 1   TCF     GOTOPOOH       V34E  TERMINATE
0680                         36,3127  1 3131 0   TCF     +2             PROCEED  CHECK FOR P42
0681    REF   2  LAST  732   36,3130  1 3123 0   TCF     REP40ALM       V32E  REDISPLAY ALARM

0682    REF  18  LAST  746   36,3131  51'455 1   INDEX   WHICH          FOR P42, ALLOW CREW TO PROCEED EVEN
0683                         36,3132  1 0014 0   TCF     14             THOUGH VEHICLE IS UNSTAGED

R0684                        ****************************************

0685                         31,2144            BANK    31
0686    REF   1              35,2000            SETLOC  P40S2
0687                         35,3747            BANK

0688    REF   1                                 COUNT*  $$/P40

0689    REF  10  LAST  743   35,3747  0 4645 1  P40AUTO  TC      MAKECADR       HELLO THERE.
0690    REF   4  LAST  474   35,3750  55'164 1            TS      TEMPR60        FOR GENERALIZED RETURN TO OTHER BANKS.
0691    REF 201  LAST  748   35,3751  0 4616 1  P40A/P    TC      BANKCALL       SUBROUTINE TO CHECK PGNCS CONTROL
0692    REF   3  LAST  518   35,3752    54255 1           CADR    G+N,AUTO       AND AUTO STABILIZATION MODES
0693    REF 222  LAST  747   35,3753  10 000 0            CCS     A              +0  INDICATES IN PGNCS, IN AUTO
0694    REF   1              35,3754  1 3766 1           TCF     TURNITON       + INDICATES NOT IN PGNCS AND/OR AUTO
0695    REF   8  LAST  736   35,3755  3 4737 0           CAF     APSFLBIT       ARE WE ON THE DESCENT STAGE?
0696    REF  13  LAST  736   35,3756  7 0106 1           MASK    FLGWRD10
0697    REF 223  LAST  748   35,3757  10 000 0           CCS     A
0698    REF   1              35,3760  1 3773 0           TCF     GOBACK         RETURN
0699    REF  31  LAST  718   35,3761  3 4747 1           CAF     BIT5           YES, CHECK FOR AUTOTHROTTLE MODE
0700                         35,3762  0 0006 1           EXTEND
0701    REF   4  LAST  475   35,3763  02 030 0           RAND    CHAN30
0702                         35,3764  0 0006 1           EXTEND
0703    REF   2  LAST  748   35,3765  1 3773 0           BZF     GOBACK
0704    REF   1              35,3766  3 3775 1  TURNITON CAF     P40A/PMD       IN AUTOTHROTTLE MODE -- RETURN
0705    REF 202  LAST  748   35,3767  0 4616 1           TC      BANKCALL       DISPLAY V50N25 R1=203  PLEASE PERFORM
0706    REF   3  LAST  513   35,3770    20623 1          CADR    GOPERF1        CHECKLIST 203  TURN ON PGNCS ETC.
0707    REF  25  LAST  748   35,3771  1 6001 1           TCF     GOTOPOOH
0708    REF   1              35,3772  1 3751 0           TCF     P40A/P         V34E  TERMINATE
0709    REF   5  LAST  748   35,3773  3 1164 0  GOBACK   CA      TEMPR60        RECYCLE
0710    REF  13  LAST  744   35,3774  0 4640 1           TC      BANKJUMP

0711                         35,3775    00203 0  P40A/PMD OCT     00203          GOODBYE.  COME AGAIN SOON.
```

But then it hit me, this was *the* code. The code that helped land human beings on the Moon in 1969. The Apollo Guidance Computer (AGC) source code, digitized and preserved in a GitHub repo by Chris Garry, based on Ron Burkey's Virtual AGC project.

What Even Is the AGC?



Astronauts manually flew Project Gemini with control sticks, but computers flew most of Project Apollo except briefly during lunar landings. Each Moon flight carried two AGCs, one each in the command module and the Apollo Lunar Module, with the exception of Apollo 7 which was an Earth orbit mission and Apollo 8 which did not need a lunar module for its lunar orbit mission.

The **Apollo Guidance Computer** was a revolutionary machine , not because of how powerful it was, but because of how *precisely engineered* it was. It had:

- 2.048 MHz processor (yes, megahertz),
- ~64 KB of memory,
- A 15-bit word length + parity bit,
- And it ran the software that guided astronauts safely to the Moon and back.

It was designed in the 1960s by the MIT Instrumentation Lab, and the code was written mostly in **AGC assembly language** by a team of engineers, many of whom were women, including the now-legendary **Margaret Hamilton**, who literally coined the term *software engineering*.

Reading this code is like reading the DNA of human ingenuity.

And I was reading it. Line by line.

I don't know how long I stared at that first file, but something shifted in me. This wasn't just code. This was history, in ASCII.

The Mindset Shift: Reading With Respect

A module that was used in the 1960s to hold 64K of memory. The module was used on the Apollo. It used paramagnetic rings, which are non-volatile, and not the modern germanium based semiconductors.

As a software engineer in 2025, I've been spoiled by tools. I write code in TypeScript with instant feedback. My IDE finishes half my sentences. I can Google any bug.

But these folks? They were writing guidance software with pencils and punch cards. On computers with 64 KB of memory. And they had to get it right on the first try, because debugging in space wasn't an option.

> *The repo itself is organized by programs used on different modules:* ***Comanche*** *for the Command Module, and* ***Luminary*** *for the Lunar Module.*

I gravitated toward the Lunar Module's software, especially the `Luminary099` build, which includes the code used during Apollo 11's actual landing.

Among all the files there, one jumped out like a spotlight on the dark side of the Moon: *`LUNAR_LANDING.agc`*

The Code That Took Us Down

```
# Page 785
                BANK    32
                SETLOC  F2DPS*32
                BANK

                EBANK=  E2DPS

#       ************************************
#       P63: THE LUNAR LANDING, BRAKING PHASE
#       ************************************

                COUNT*  $$/P63

P63LM           TC      PHASCHNG
                OCT     04024

                TC      BANKCALL        # DO IMU STATUS CHECK ROUTINE R02
                CADR    R02BOTH

                CAF     P63ADRES        # INITIALIZE WHICH FOR BURNBABY
                TS      WHICH

                CAF     DPSTHRSH        # INITIALIZE DVMON
                TS      DVTHRUSH
                CAF     FOUR
                TS      DVCNTR

                CS      ONE             # INITIALIZE WCHPHASE AND FLPASS0
                TS      WCHPHASE

                CA      ZERO
                TS      FLPASS0

                CS      BIT14
                EXTEND
                WAND    CHAN12          # REMOVE TRACK-ENABLE DISCRETE.
```

LUNAR_LANDING.agc Code File

`LUNAR_LANDING.agc` is exactly what it sounds like , it's the logic that executed the **Powered Descent Phase** of the lunar landing.

This is the moment where the Lunar Module, called **Eagle**, detached from the Command Module, rotated, fired its descent engine, and began that iconic drift toward the surface of the Moon.

Every pulse of the engine. Every adjustment in velocity. Every moment that kept Armstrong and Aldrin from crashing into a crater, it happened under the watch of this file.

Let's explore a few segments from the file and unpack what they're doing, in plain English:

1. Starting the Descent — `P63`

```
CALL    P63
```

This command calls **Program 63** : the designated program for Powered Descent. It's like activating a whole subroutine dedicated to calculating how to control the engine, thrust, pitch, and descent profile.

Program 63 begins running a continuous loop of feedback between sensor input and engine control, adapting based on real-time data from the radar, gyros, and inertial systems.

> *In today's terms? Imagine a self-flying drone landing on a moving platform using nothing but a calculator.*

2. Velocity Vectors — `DXCH DESVX , DESVZ`

```
DXCH    DESVX
DXCH    DESVZ
```

These lines are swapping values into registers that hold the **descent velocities :** horizontal (X) and vertical (Z).

This logic checks:

- How fast are we falling?
- Are we drifting sideways?
- Do we need to correct?

The AGC didn't have floating-point arithmetic or Kalman filters, it was all integer math, but incredibly accurate due to smart use of scaling and units.

3. Radar Integration — `TCF RADARCHECK`

```
TCF       RADARCHECK
```

This is the conditional transfer (a jump) to a section where the system checks the **landing radar.**

Before switching fully to surface-relative navigation, the AGC had to confirm:

- Radar lock is established,
- Altitude is updating correctly,
- And the data is stable enough to rely on.

One of the fun facts? During Apollo 11, the radar was “spamming” the AGC with unexpected interrupts, which caused the famous `1202` and `1201` alarms. The AGC : thanks to its smart task scheduling, handled it gracefully and didn’t crash.

4. Abort Scenarios — `TCF ABORT`

```
TCF     ABORT
```

One of the most important safety nets in this file is the abort procedure.

If the landing trajectory becomes unsafe, due to fuel, velocity, or navigation failure, this code triggers an immediate abort, cutting descent and initiating ascent engine burn to escape lunar gravity and return to orbit.

It was life insurance written in assembly.

What I Realized While Reading

Reading `LUNAR_LANDING.agc` wasn’t just technical. It was philosophical.

Here’s what I took away:

1. Simplicity is not weakness

This code is lean, every line has purpose. There’s no abstraction for the sake of it. It’s a refreshing reminder that *clarity beats cleverness* — especially when

lives are on the line.

2. Constraints breed brilliance

They wrote software that worked in 64KB of memory. Today, some NPM packages eat that before you even import a function. These constraints forced creativity, not workarounds, but elegant designs.

3. Engineering is about trust

When Armstrong took manual control of the Lunar Module, he still relied on AGC to hold him steady. That level of trust in a computer, in 1969, speaks volumes about the discipline and testing behind this code.

Historical Context You Can Feel

Margaret Hamilton once said:

> *"There was no second chance. We all knew that."*

You can *feel* that in the code.

There are jokes in the comments, yes, small human touches like:

```
# THE LAST THING THE LM PILOT WILL EVER SEE IF THIS PROGRAM FAILS
```

But there's also discipline. Every label, every routine is carefully architected. There's no wiggle room for experimentation, it either worked or it didn't.

And that pressure molded some of the most reliable software ever written!

Software Development | Programming | Coding | Code | Moon Landing

Published in The Deep Hub

1.6K Followers · Last published 1 day ago

Follow

Your data science hub. A Medium publication dedicated to exchanging ideas and empowering your knowledge.

Written by Palash Mishra

548 Followers · 143 Following

Follow

Experienced FullStack Engineer and Machine Learning enthusiast with a passion for developing innovative web products and optimizing ML solutions.

Responses (13)

A. Calderwood

What are your thoughts?

Loshan he/him
Apr 9

> This wasn't just code. This was history, in ASCII.

Wow, I felt that! 😲 You didn't just read code — you touched a piece of space history. I love how you made me see code as something more than just tech... it's human achievement in every line! 🚀📜✨

63 1 reply Reply

Wallace Mann
4 days ago

Thanks! That was cool history.

I interned at the company building the Saturn V first stage flight control computer. It was the size of a garbage can and built entirely from discrete RLCT components.

57 Reply

Ernst Reidinga
Apr 9

Great article! Couldn't stop reading 😁 immediately starred the repo, I know what to do this weekend ❤😁

53 Reply

See all responses

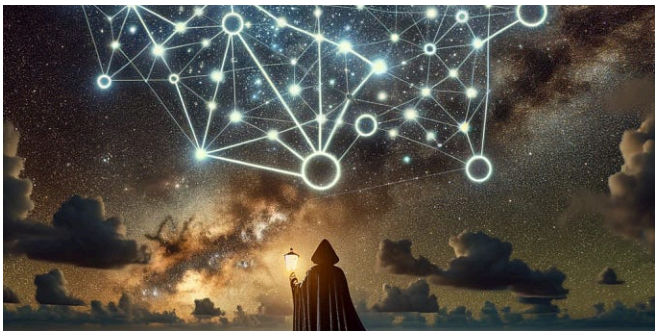
More from Palash Mishra and The Deep Hub

In The Deep Hub by Palash Mishra

Getting Started with PyTorch: A Beginner-Friendly Guide

If you've ever wondered how to build and train deep learning models, PyTorch is one of the...

Dec 3, 2024 55

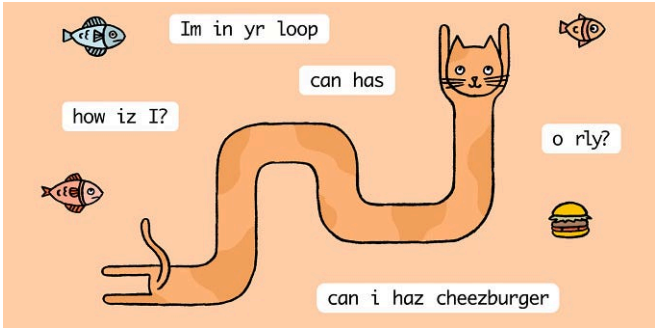


In The Deep Hub by Jorgecardete

Convolutional Neural Networks: A Comprehensive Guide

Exploring the power of CNNs in image analysis

Feb 7, 2024 2.8K 39

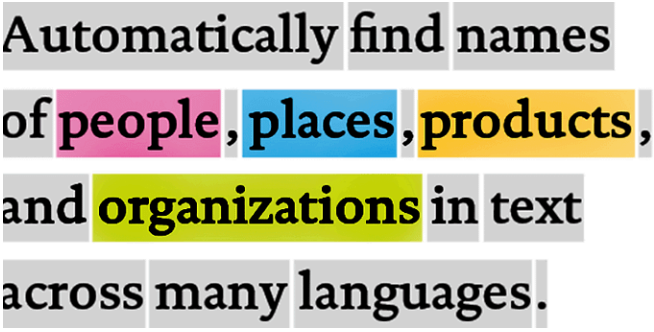


In The Deep Hub by Kiran Maan

I Found an Amazing Programming Language

Previously, I never heard about this language...

Mar 20 347 8



Palash Mishra

Simple Use Case : Building a Named Entity Recognition Syste...

Named Entity Recognition (NER) is a crucial task in Natural Language Processing (NLP)...

Dec 29, 2023 21

See all from Palash Mishra | See all from The Deep Hub

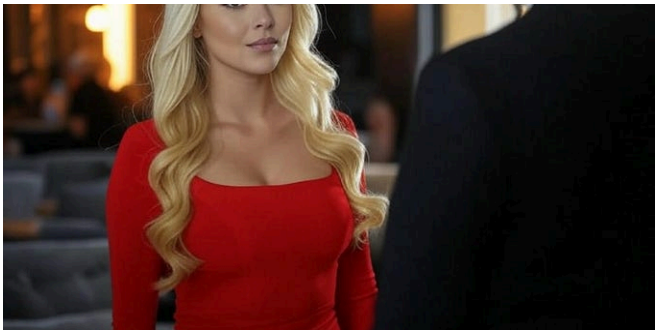
Recommended from Medium

Ignacio de Gregorio

Llama 4 is Out. Meta & the US Are in Big Trouble.

Meta recently unveiled its latest series of AI models, Llama 4, allegedly introducing...

6d ago 1.9K 44



Laura Hart

Top 5 Things Women Instantly Check Out in a Man

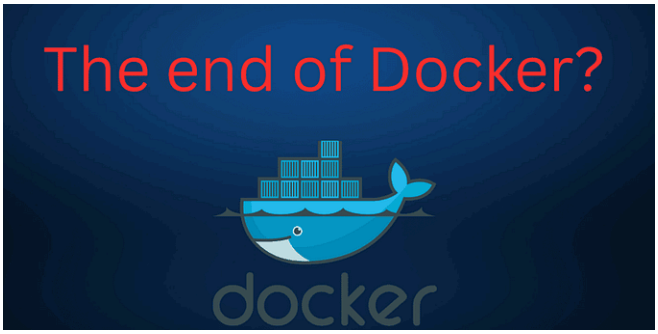
Think she's only checking out your shoes? Think again. Here's what actually makes that...

Apr 6 3.2K 88

In Internet of Technology by Jules May

Error handling: We're doing it all wrong

How decades of exception-throwing have been making our programs less stable, and...



Devlink Tips

The end of Docker? The Reasons Behind Developers Changing The...

Docker once led the container revolution—but times have changed. Developers are...

Apr 11 249 14

Mar 20 844 28

In TechToFreedom by Yang Zhou

11 Outdated Python Modules That You Should Never Use Again

And their modern alternatives that you should master

Apr 11 757 18

Alex Dunlop

It took us 2 months to adopt cursor and 1 day to adopt Augmentcode

I am currently working at an AI startup in Bank, London. Here is a photo of the view...

Apr 3 128 4

See more recommendations