IMPLEMENTATION DETAIL

Converting Fixed point to floating point format and vice versa.

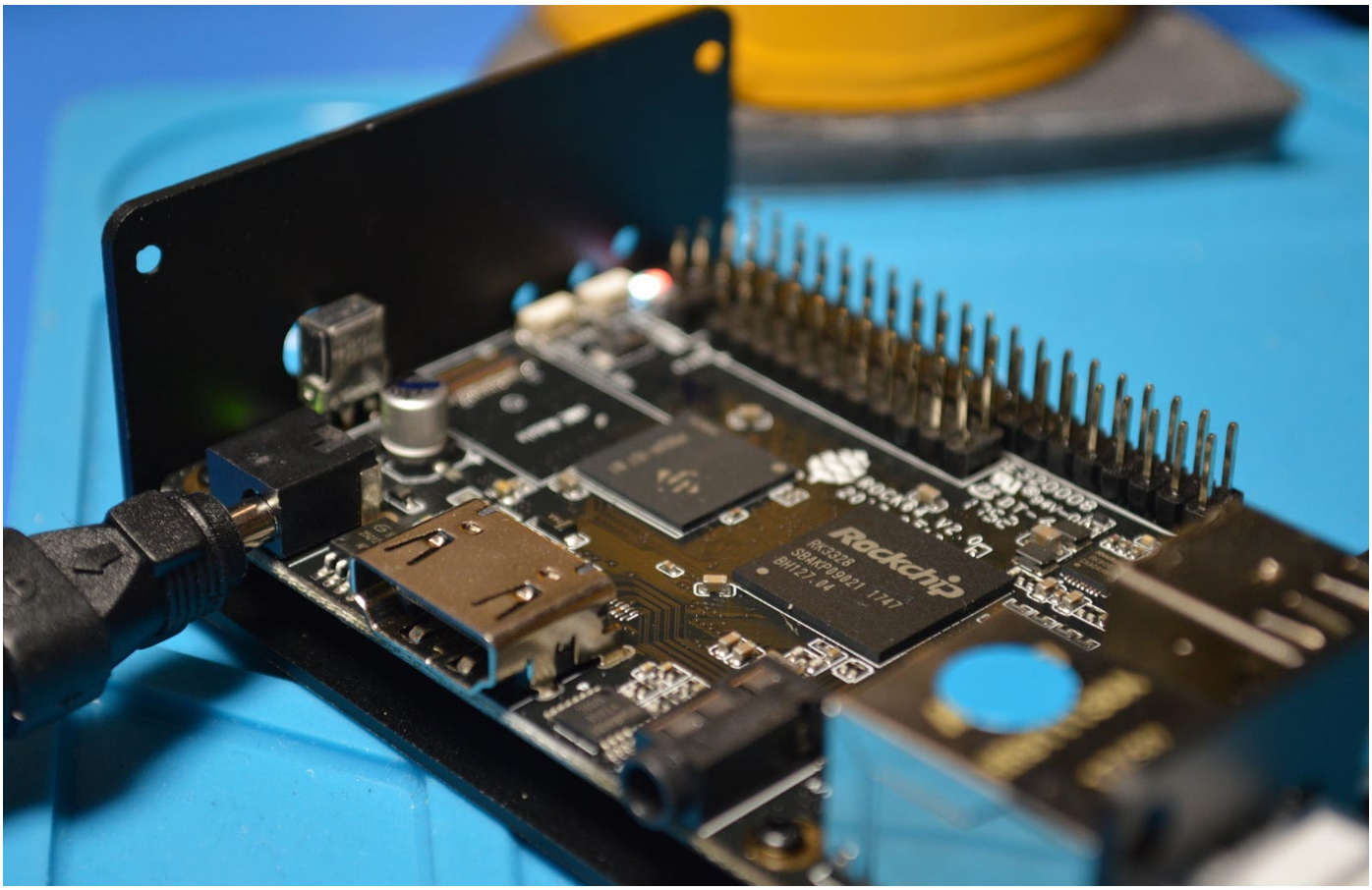
Dealing with floating and fixed-point representation of real numbers

Arunkumar Maniam Rajan · Follow

Published in Incredible Coder

5 min read · Nov 28, 2019

Listen Share



decimal data and process it, more likely we are going to bounce into floating and fixed-point conversion.

In most hardware, the data of some signal or a quantity is stored in a format called “Q” fixed point format. The embedded systems usually wouldn’t have the complexity of storing decimals in floating-point format. They store numbers in the registers mostly 16 bits to 32 bits.

In places where Arithmetic and Logical unit can handle only fixed-point arithmetic, we will not have the luxury of using floating-point numbers.

So there would be a need to convert from floating point to fixed point and vice versa when storing and retrieving decimal values from embedded systems that store the data at fixed length memory locations or registers. (We also have the option of performing fixed-point arithmetic but we need to handle special cases, for example, saturation and stuff while performing multiplication.)

The general workflow would be

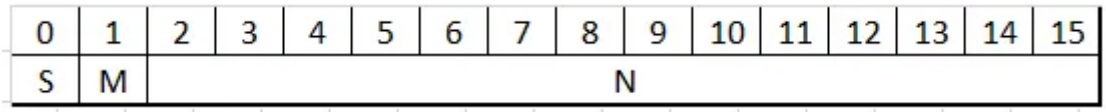
1. Read data from the registers in fixed-point format.
2. Convert to floating-point
3. Perform floating-point arithmetic to process the data.
4. Convert the result back to the fixed point format.
5. Write the result back to the registers

Before we dive into the actual conversion itself we can take a look at what is “Q” format.

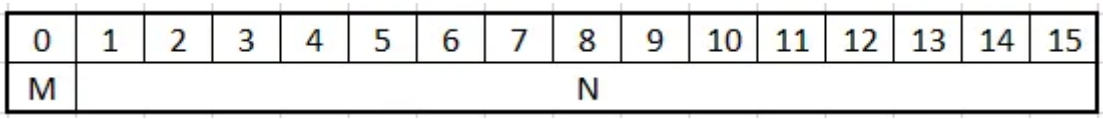
Q(m.n) represents there would be m bits before the decimal point and n bits after the decimal point.

E.g. For a system with 16 bits length Q(1.15) means there would be one bit before the decimal point(magnitude) and 15 bits to denote the numbers after the decimal

There also signed and unsigned version of “Q” format. “Unsigned Q” format or “UQ” means that all the bits are used to denote the value of the number, while in signed “Q” format one bit is used to denote sign rest of the bits would be used to denote the magnitude of the number.



Signed Q1.14 1 bit for sign, 1 bit for M and 14 bits for N



Unsigned Q1.15 1 bit for M and 15 bits for N

Notice that it’s up to the implementation to decide the value of M and N. M and N are decided by three factors

1. Whether signed values are needed.
2. Range of values in the system.
3. Resolution of values required.

While the sign is obvious, let's see how M and N affect range and resolution of a system.

Let’s consider the example of a fixed width of 16 bits. In the 16 bits, we can have different formats like Q1.14, Q15.1, Q13.3, UQ1.15 etc. For comparison, we could just take UQ1.15 and UQ15.1

With UQ1.15 fixed-point format, the range of numbers we could denote numbers from 0 to 1.99996948242. Know that 1.99996948242 is floating-point the equivalent fixed-point value is 65535 which is the bit pattern of all 1s in the 16-bit memory location or register. The resolution would be 0.00003051 which is determined by the “N”. This is nothing but the amount of value change in floating-point when the least significant bit changes in fixed-point

Below images show the range and resolution of “Qm.n” formats

Qm.n format

$$\text{Range} : [-2^{m-1}, 2^{m-1} - 2^{-n}]$$

$$\text{Resolution}: [-2^{-n}]$$

Signed Qm.n

UQm.n format

$$\text{Range} : [0, 2^{m} - 2^{-n}]$$

$$\text{Resolution}: [-2^{-n}]$$

Unsigned Qm.n

So if there is a need to represent a large range we would increase m at the expense of decreased resolution. If the resolution is important, we would have Q format with large N., For example, unsigned Q15.1 format would have a range of 0 to 32767.5 and resolution of 0.5. Unsigned Q1.15 format would have a range of 0 to 1.99996 and a resolution of 0.00003051.

Converting from fixed-point to floating-point

Let X be the fixed-point number to convert it into floating-point

1. Convert the fixed-point number as an integer.
2. Divide the number by 2^n (2 to the power of n).

Below is the python snippet.

```
"""
x is the input fixed number which is of integer datatype
e is the number of fractional bits for example in Q1.15 e = 15
"""
def to_float(x,e):
    c = abs(x)
    sign = 1
    if x < 0:
        # convert back from two's complement
        c = x - 1
        c = ~c
        sign = -1
    f = (1.0 * c) / (2 ** e)
    f = f * sign
    return f
```

fixed_to_float.py hosted with ❤️ by GitHub view raw

https://gist.github.com/pragmaticarun/0270e8037d6607b17ef770a672873c8a

Converting from floating-point to fixed-point

Let F be the floating-point number to convert it to the fixed point number

1. Multiply it by 2^n
2. Round the value to the nearest integer
3. If F is negative take two's complement of the value arrived at step 2.

Below is the python script

```
"""
f is the input floating point number
e is the number of fractional bits in the Q format.
    Example in Q1.15 format e = 15
"""
def to_fixed(f,e):
```

```
        # next three lines turns b into it's 2's complement.
        b = abs(b)
        b = ~b
        b = b + 1
    return b
```

float_to_fixed.py hosted with ❤ by GitHub view raw

https://gist.github.com/pragmaticarun/0751416405d34943a281091b3cb103bf

A word about the convention, Q1.15 means, it is a signed fixed-point. Care must be taken to confirm that implementation has signed bit is included in magnitude or excluded from it. UQ1.15 means, it is an unsigned fixed-point number.

By the way, if you are wondering why the formats get the name of fixed and floating, it because of the nature of the position of the decimal point. In fixed the number of bits denoting the magnitude(exponent) and a fraction (mantissa) is independent of the value it is denoting. While in floating-point the position of the decimal point depends on the value of the real number that it is representing.

Also if you notice the resolution also known as the precision of a Q format is fixed since it only depends on the number of fractional bits.

Below is an example conversion table in signed Q1.15 format.

Floating point	Fixed point
0.999969482	32767
-1	-32768
1	out of range
0	0
0.9999	32765
0.00003051	1
-0.00003051	-1

Fixed-point to the floating-point conversion table.

Note: If you are wondering why store negative numbers in two's complement notation, the short answer is it takes one instruction to deal with adding two numbers with a

A few wiki links that will help to understand the formats better.

Fixed-point arithmetic

In computing, a fixed-point number representation is a real data type for a number that has a fixed number of digits...

en.wikipedia.org

Two's complement

Two's complement is a mathematical operation on binary numbers and is an example of a radix complement. It is used in...

en.wikipedia.org

Floating-point arithmetic

In computing, floating-point arithmetic (FP) is arithmetic using the formulaic representation of real numbers as an...

en.wikipedia.org

Hope it was useful.

Cheers and thanks for reading.

Programming Python Fixed Floating Software Engineering

Follow

Written by Arunkumar Maniam Rajan

33 Followers · Editor for Incredible Coder

Son | Student | Friend | Programmer | Husband | Father | Reader | Listener | Writer | Problem Solver | Human (Super)

More from Arunkumar Maniam Rajan and Incredible Coder

Arunkumar Maniam Rajan in Incredible Coder

A programmer's first day on Codeforces.

Although I registered on code forces some 3 weeks back, today I got some time to take Codeforces for a ride.

3 min read · Nov 21, 2019

Arunkumar Maniam Rajan in Incredible Coder

Newbie's C++ competitive programming template.

A template get started with C++ for competitive programming

4 min read · Nov 28, 2019

Arunkumar Maniam Rajan

The Power of Half-Smile.

A famous and central theme in the circle of mindful thinking, this article throws light into how this powerful tool can be used to give us...

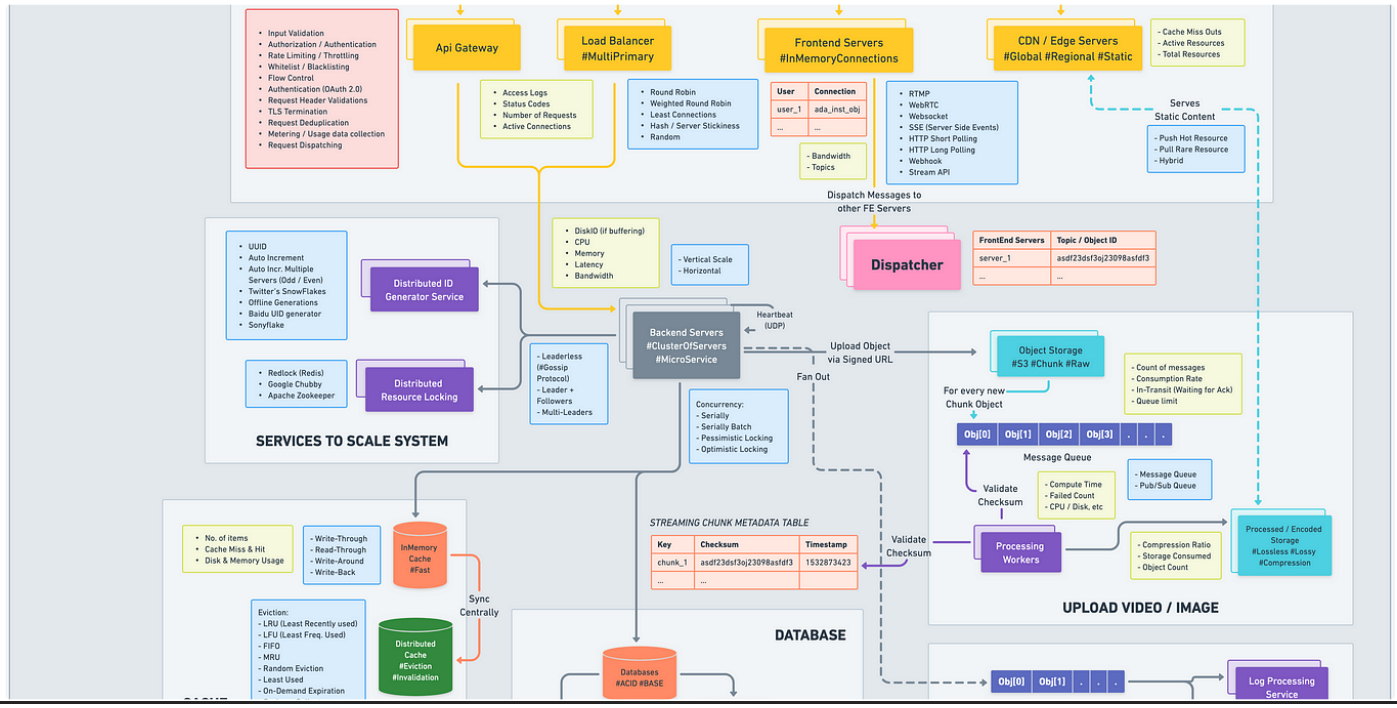
3 min read · Dec 6, 2019

60

See all from Arunkumar Maniam Rajan

See all from Incredible Coder

Recommended from Medium

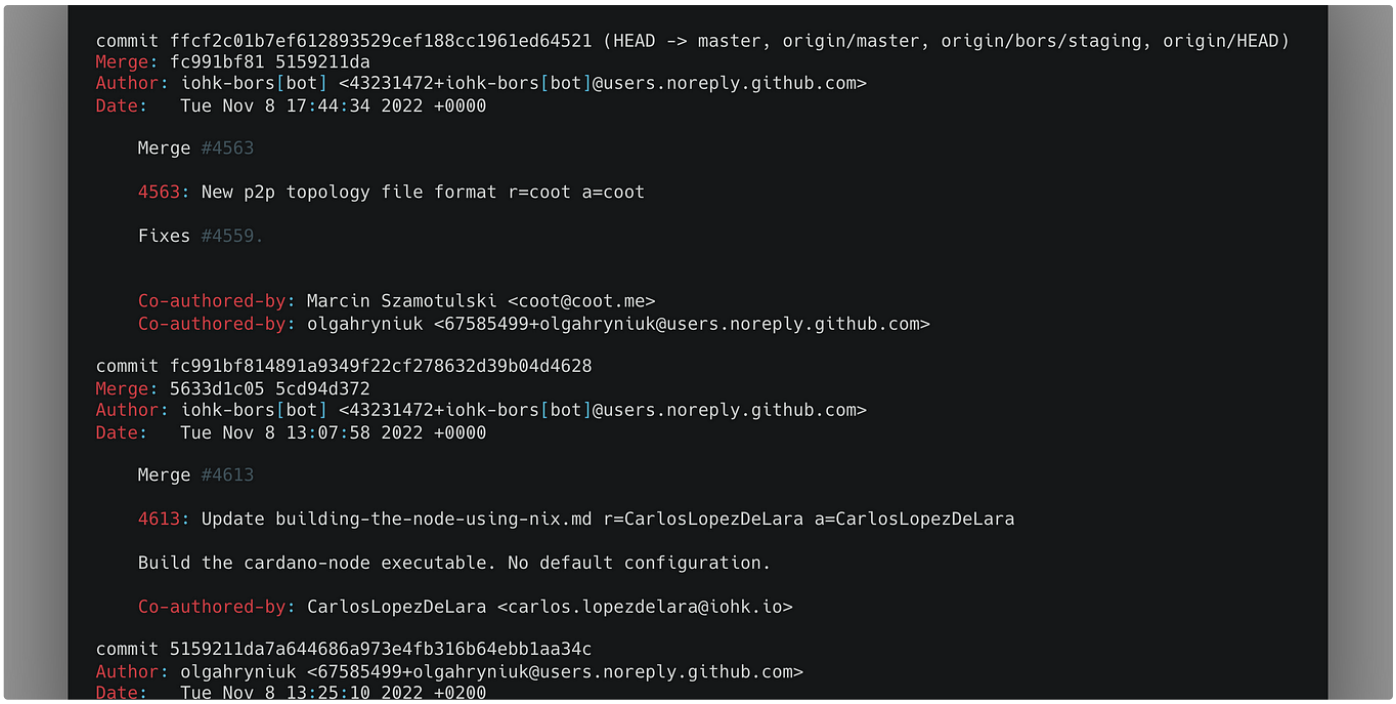


System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✦ · 9 min read · Apr 20

5.8K 48



Jacob Bennett in Level Up Coding

Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

✦ · 4 min read · Nov 14, 2022

7.2K 75

Lists

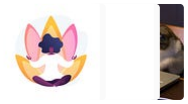
General Coding Knowledge

Coding & Development

11 stories · 43 saves

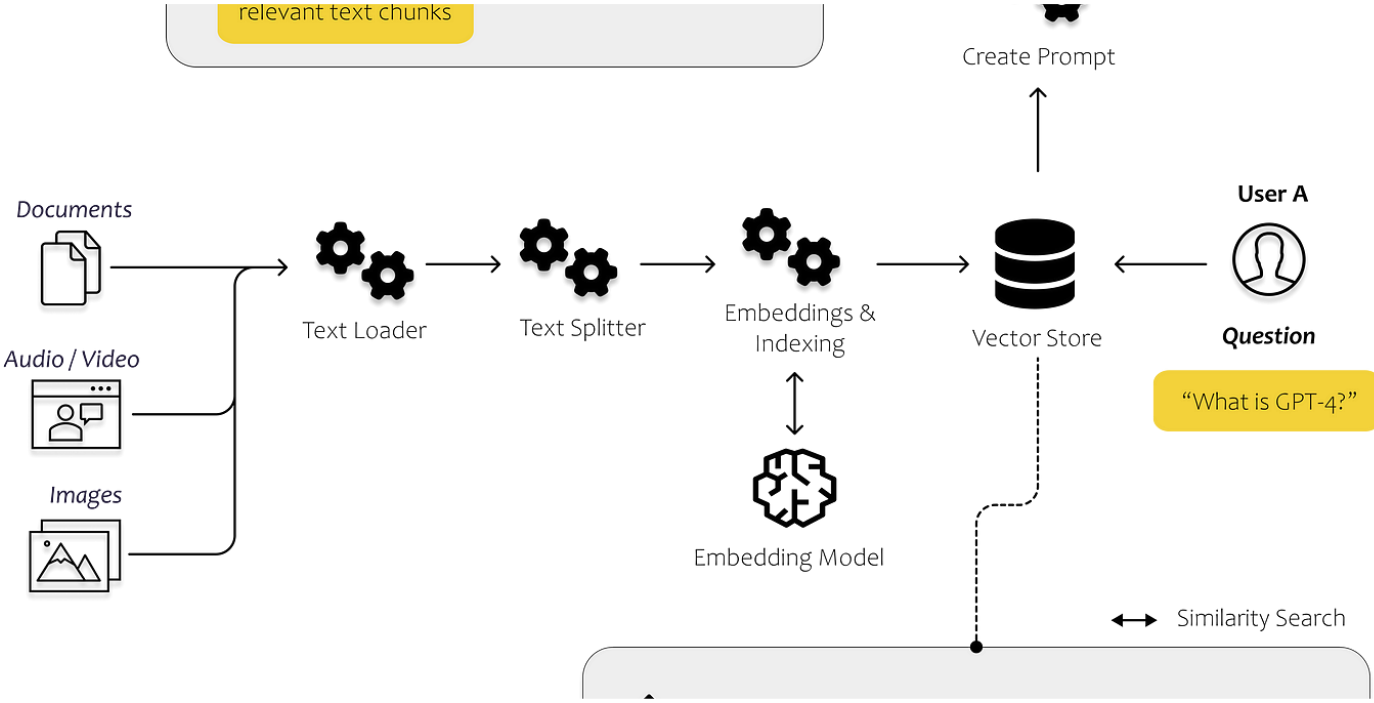
It's never too late or early to start something

10 stories · 23 saves



Stories to Help You Grow as a Software Developer

19 stories · 168 saves



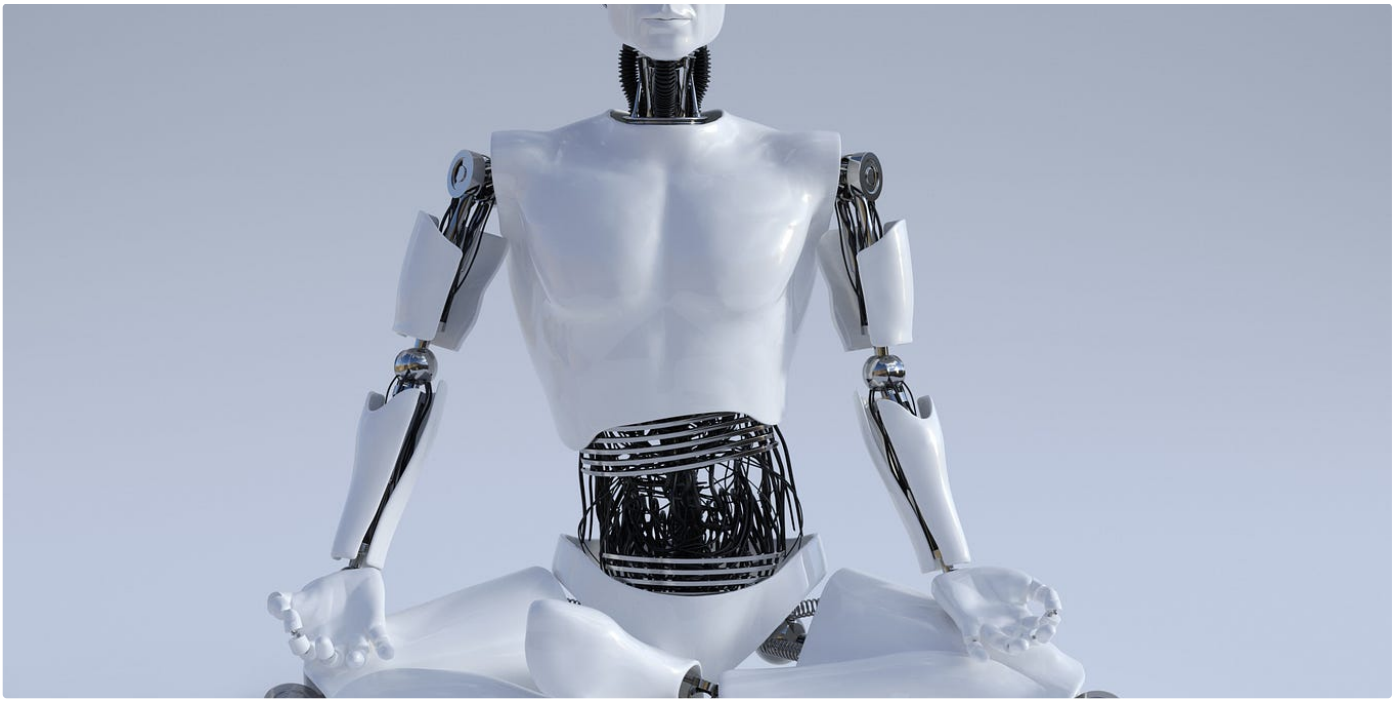
Dominik Polzer in Towards Data Science

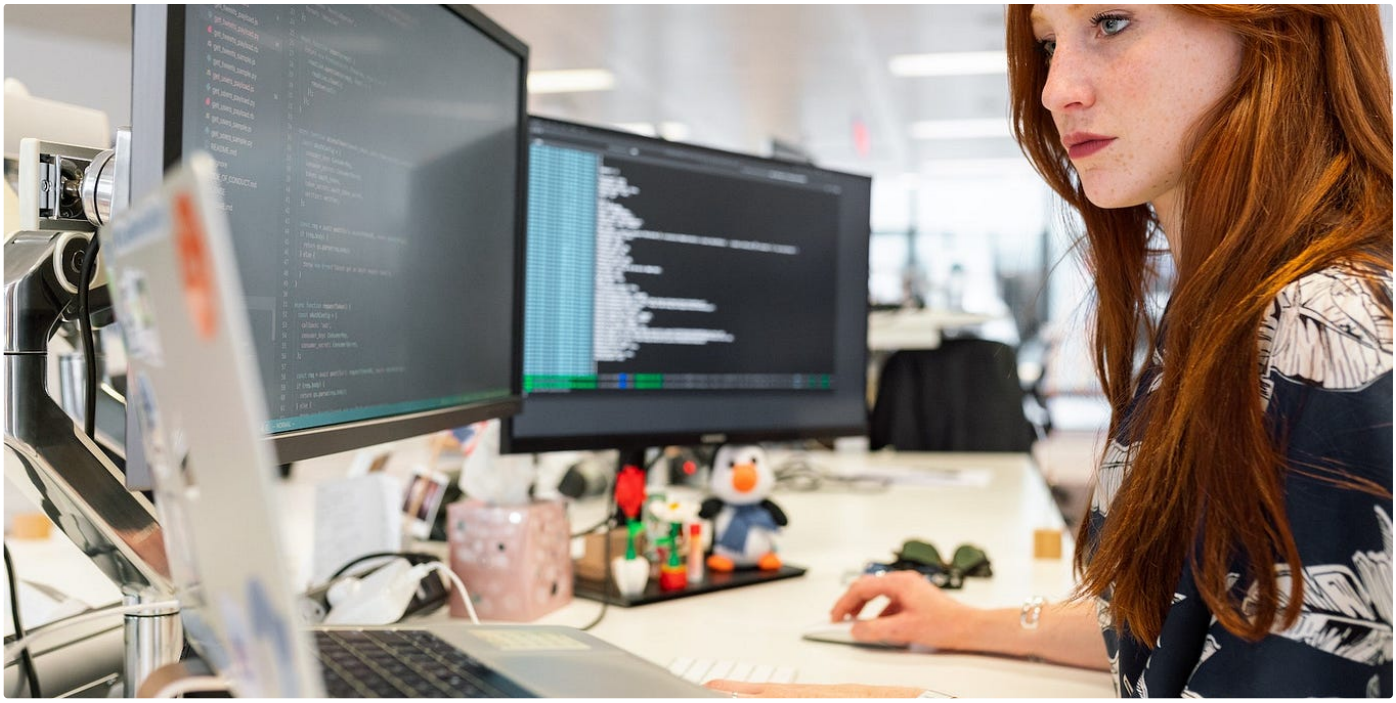
All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

· 25 min read · Jun 21

1.8K 18





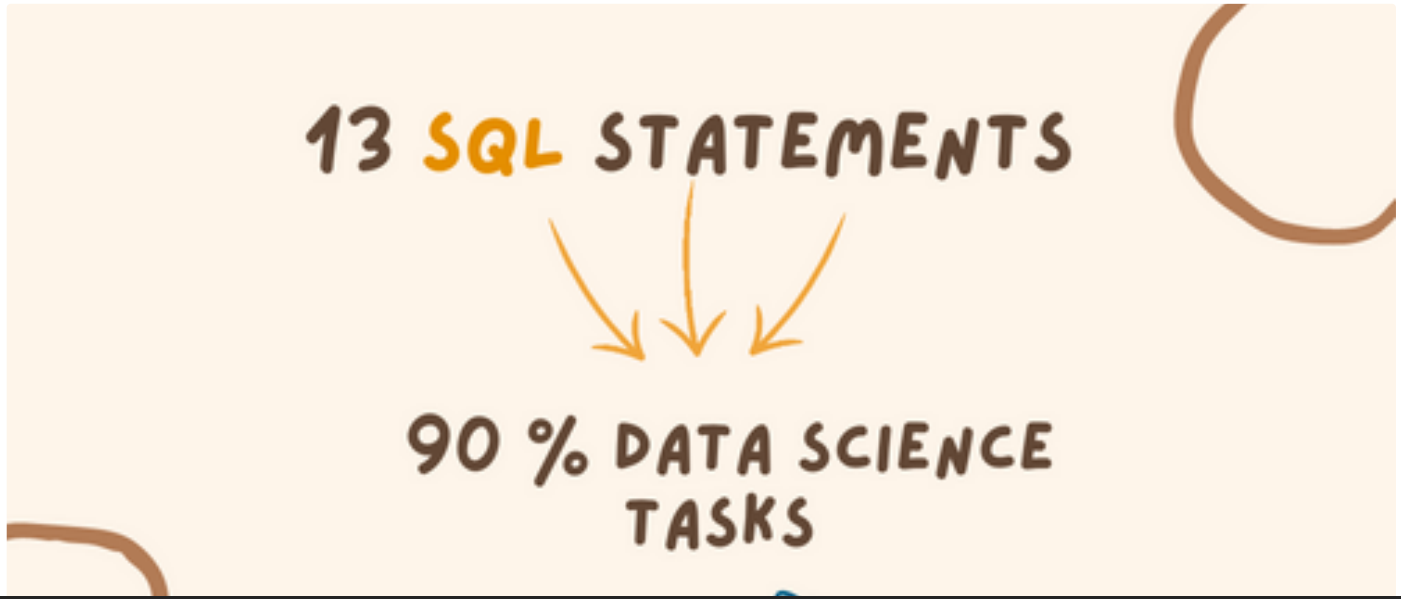
The Coding Diaries in The Coding Diaries

Why Experienced Programmers Fail Coding Interviews

A friend of mine recently joined a FAANG company as an engineering manager, and found themselves in the position of recruiting for...

· 5 min read · Nov 2, 2022

4.8K 106



Youssef Hosni in Level Up Coding

13 SQL Statements for 90% of Your Data Science Tasks

Structured Query Language (SQL) is a programming language designed for managing and manipulating relational databases. It is widely used by...

✨ · 15 min read · Feb 26

2.9K 33