

Accidentally building a Neural Network

A Ruby product recommendation journey

Tiny Ruby Conf, Oct 1st 2026

Hi, I'm Nicolò! 🖐️

Nicolò Rebughini

github.com/nirebu

Currently developer at Fuzu

(after spending a decade in cinema post-production)



Continua come Nicolò²



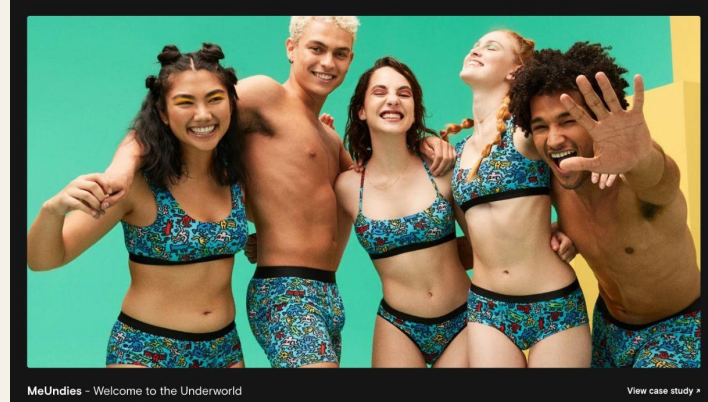
What I worked on at Nebulab



Cometeer - A New Day Has Arrived on Earth for Coffee
[View case study »](#)



Framework - The World's First Modular Laptop
[View case study »](#)



MeUndies - Welcome to the Underworld

[View case study »](#)



DMC - French Quality Since 1746

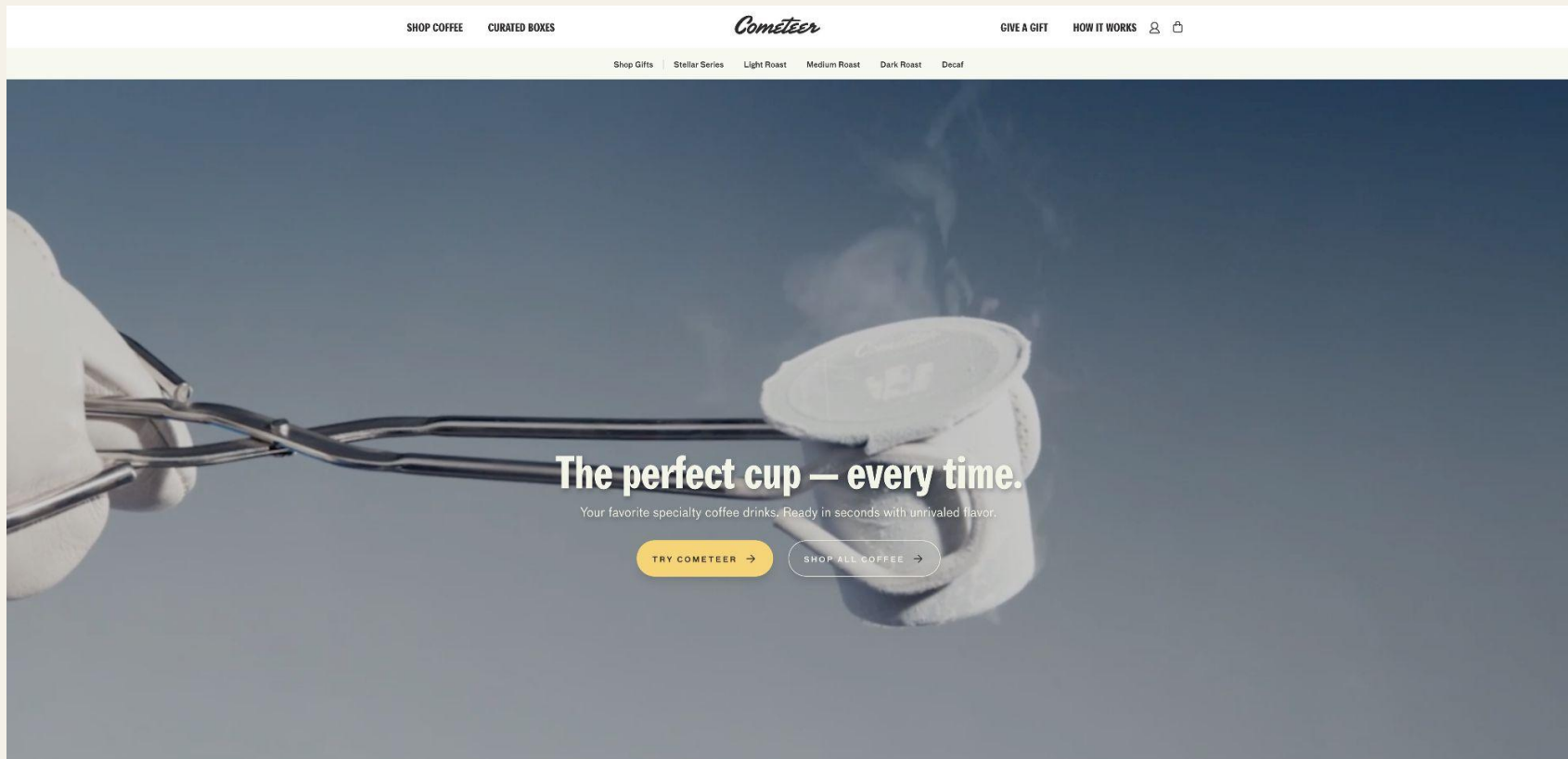
[View case study »](#)



Huckberry - Gear for Today, Inspiration for Tomorrow
[View case study »](#)

Maisonette - Boutique Clothes & Decor for Kids & Babies
[View case study »](#)

Cometeer



So many choices...

March Featured Picks

From chocolate-forward comfort to radiant citrus and honeyed sweetness, each brew is designed to refresh your ritual and awaken your palate.



SQUARE MILE
Menser Abo Hika
Creamy, fruit-layered washed Ethiopian coffee, with notes of blackberry and chocolate.
[Learn More](#)

🌍	Ethiopia
🔥	Light Roast
🍷	Bright & Fruity

\$22 \$2.75 per cup

ADD



WINSLOW COFFEE CO.
Steady Hand Half Caff
Steady, rich half-caff blend from Colombia and Brazil, with notes of raisin, dark caramel, and chocolate.
[Learn More](#)

🌍	Colombia & Brazil
🔥	Dark Roast
🍷	Chocolatey & Sweet

\$18.49 \$2.31 per cup

ADD



PEAK SEASON
Wake Up Call
Balanced, breakfast-ready blend from Guatemala and Ethiopia, with notes of dried fruit, milk chocolate, and molasses.
[Learn More](#)

🌍	Guatemala & Ethiopia
🔥	Medium Roast
🍷	Chocolatey & Caramelized

\$18.49 \$2.31 per cup

ADD



GEORGE HOWELL
Santa Lucia
Sweet and balanced Brazilian pulped natural, with notes of caramel, almond, and cherry.
[Learn More](#)

🌍	Brazil
🔥	Light Roast
🍷	Sweet & Nutty

\$22 \$2.75 per cup

ADD



PEAK SEASON
Room Service
Smooth, indulgent dark blend from Costa Rica and Kenya, with notes of berry, caramel, and chocolate.
[Learn More](#)

🌍	Costa Rica & Kenya
🔥	Dark Roast
🍷	Chocolatey & Caramelized

\$17.49 \$2.18 per cup

ADD



EQUATOR
Kenya Lenana
An exceptional Kenyan coffees, fruity and clean.
[Learn More](#)

🌍	Kenya
🔥	Light Roast
🍷	Bright & Floral

\$19.99 \$2.49 per cup

ADD



LINGO
La Victoria
Lush, fruit-sweet Colombian single origin, with notes of cherry, candied apple, and caramel.
[Learn More](#)

🌍	Colombia
🔥	Medium Roast
🍷	Sweet & Fruity

\$25 \$3.12 per cup

ADD



GO GET EM TIGER
Chelbesa
With notes of pear, jasmine, and papaya.
[Learn More](#)

🌍	Guj, Oromia, Ethiopia
🔥	Light Roast
🍷	Bright & Floral

\$19.99 \$2.49 per cup

ADD



GRAVITY WAVE
Drift Decaf
Energizing Colombian EA decaf, with notes of milk chocolate, almond, and cherry.
[Learn More](#)

🌍	Colombia
🔥	Medium Roast
🍷	Sweet & Nutty

\$17.49 \$2.18 per cup

ADD



LINGO
Duromina
Washed Ethiopian landrace from Agaro, with notes of lavender, lemon drop, and blackcurrant.
[Learn More](#)

🌍	Ethiopia
🔥	Light Roast
🍷	Sweet & Fruity

\$18.49 \$2.31 per cup

ADD

Premium Coffees

Top picks from our Select and Stellar series lines, featuring flavorful and unique single estate coffees from around the world.



LINGO
La Victoria
Lush, fruit-sweet Colombian single origin, with notes of cherry, candied apple, and caramel.
[Learn More](#)

🌍	Colombia
🔥	Medium Roast
🍷	Sweet & Fruity

\$25 \$3.12 per cup

ADD



LINGO
Finca La Concepción
Lush, fruit-forward cup from Narillo, Colombia, with notes of honey, cherry, and wine-like acidity.
[Learn More](#)

🌍	Colombia
🔥	Light Roast
🍷	Sweet & Fruity

\$25 \$3.12 per cup

ADD



SWEET TALKER
Brightside
Juicy, expressive Kenyan washed coffee, with notes of black currant, red berry, and a wine-like acidity.
[Learn More](#)

🌍	Kenya
🔥	Light Roast
🍷	Juicy & Caramelized

\$25 \$3.12 per cup

ADD



PROUD MARY
Los Pinos
Elegant Nicaraguan natural, with notes of brown spice, milk chocolate, and stone fruit.
[Learn More](#)

🌍	Nicaragua
🔥	Light Roast
🍷	Fruity & Sweet

\$25 \$3.12 per cup

ADD



BIRD ROCK
Miroroma AA
Lively washed Kenyan AA, with notes of blackberry jam, molasses, dried mango, and hibiscus.
[Learn More](#)

🌍	Kenya
🔥	Light Roast
🍷	Sweet & Fruity

\$25 \$3.12 per cup

ADD



TOP PICK



GO GET EM TIGER



Stellar Series



HACHI



GO GET EM TIGER

Omakase bundles

Omakase offering
of **Curated Boxes**
to overcome
decision paralysis



Mixed Roast Box

Explore the full spectrum of coffee flavor with our Mixed Roast Box. A curated selection of light, medium, and dark roasts for the adventurous palate.

ROAST LEVEL Mixed

Mixed

Light

Medium

Dark

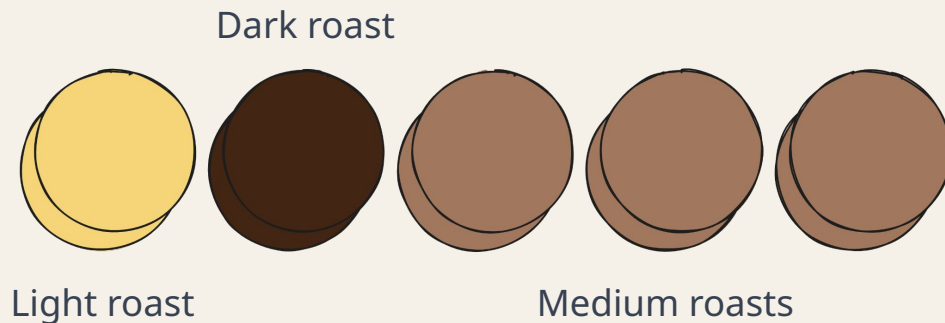
Decaf



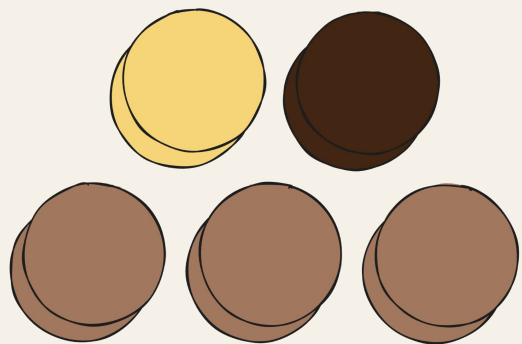
Looking for something specific? [Build Your Own Box!](#)

Curated bundle anatomy

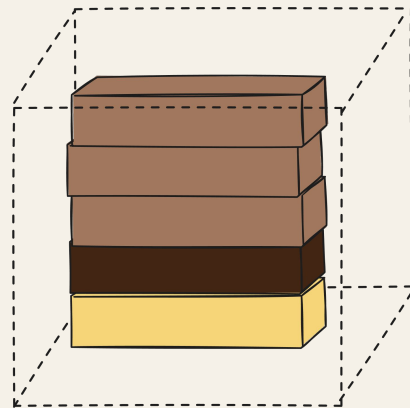
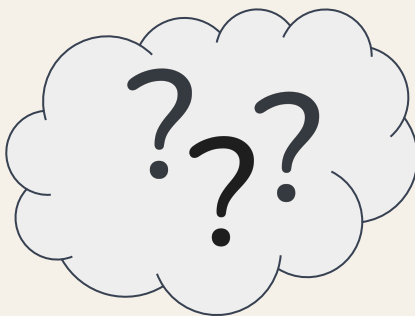
Inside the **Mixed Roast Box** we find 5 slots, each calling for **specific product requirements**.



Androids dream of electric coffee, humans don't



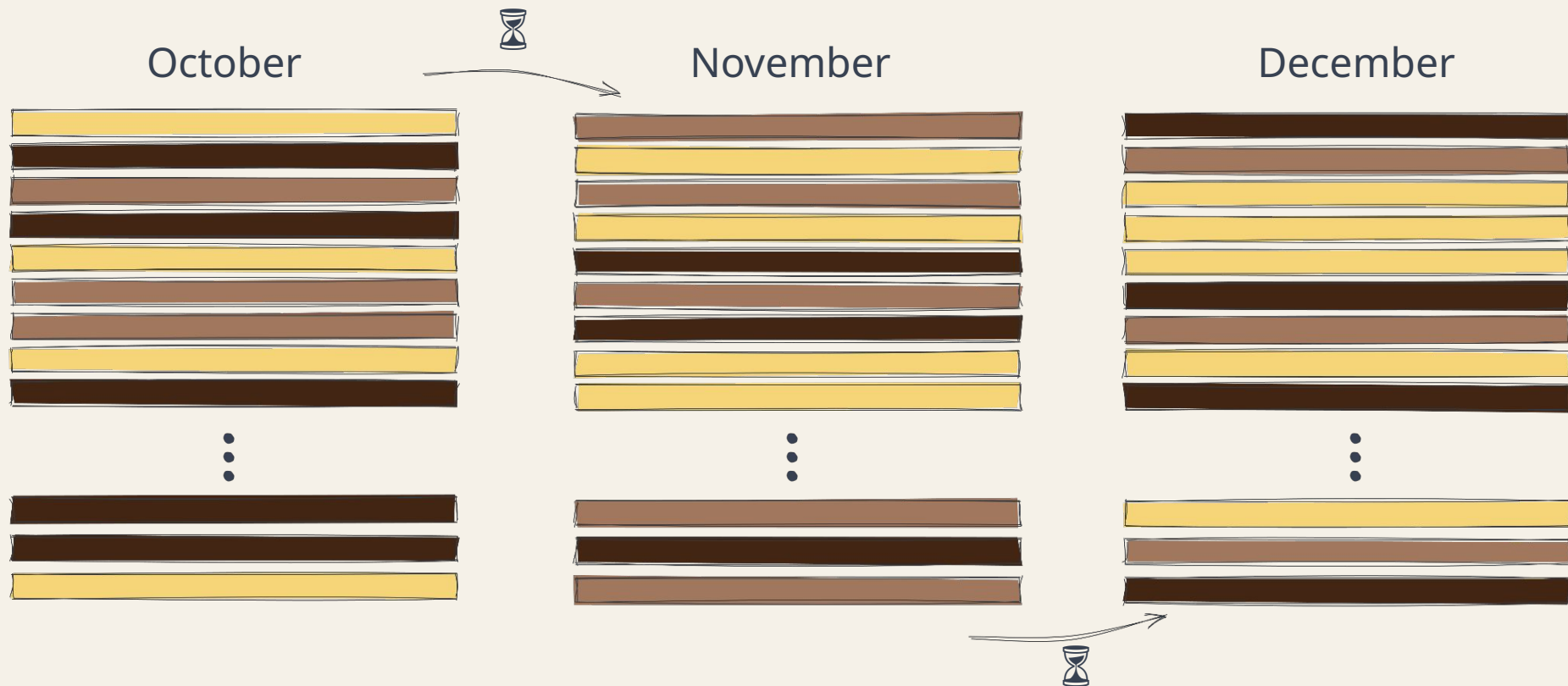
What the customer buys:
5 virtual products



What the warehouse ships:
5 tangible SKUs

How do we make the question mark
feel like a barista?

Manually curated collections



A simple collection mapper

```
def map!  
  @products.each do |product|  
    @requirement_sets.each_with_index do |requirement_set, index|  
      next unless empty_slot?(index)  
      next unless requirement_set.matches?(product)  
      eligible_variant = product.default_variant  
      next unless enough_inventory?(eligible_variant, @mapped_variants)  
  
      @mapped_variants[index] = eligible_variant  
      break if @mapped_variants.compact.count == @requirement_sets.count  
    end  
  end  
end
```

A simple collection mapper

```
def map!  
  @products.each do |product|  
    @requirement_sets.each_with_index do |requirement_set, index|  
      next unless empty_slot?(index)  
      next unless requirement_set.matches?(product)  
      eligible_variant = product.default_variant  
      next unless enough_inventory?(eligible_variant, @mapped_variants)  
  
      @mapped_variants[index] = eligible_variant  
      break if @mapped_variants.compact.count == @requirement_sets.count  
    end  
  end  
end
```

A simple collection mapper

```
def map!  
  @products.each do |product|  
    @requirement_sets.each_with_index do |requirement_set, index|  
      next unless empty_slot?(index)  
      next unless requirement_set.matches?(product)  
      eligible_variant = product.default_variant  
      next unless enough_inventory?(eligible_variant, @mapped_variants)  
  
      @mapped_variants[index] = eligible_variant  
      break if @mapped_variants.compact.count == @requirement_sets.count  
    end  
  end  
end
```

A simple collection mapper

```
def map!  
  @products.each do |product|  
    @requirement_sets.each_with_index do |requirement_set, index|  
      next unless empty_slot?(index)  
      next unless requirement_set.matches?(product)  
      eligible_variant = product.default_variant  
      next unless enough_inventory?(eligible_variant, @mapped_variants)  
  
      @mapped_variants[index] = eligible_variant  
      break if @mapped_variants.compact.count == @requirement_sets.count  
    end  
  end  
end
```

A simple collection mapper

```
def map!  
  @products.each do |product|  
    @requirement_sets.each_with_index do |requirement_set, index|  
      next unless empty_slot?(index)  
      next unless requirement_set.matches?(product)  
      eligible_variant = product.default_variant  
      next unless enough_inventory?(eligible_variant, @mapped_variants)  
  
      @mapped_variants[index] = eligible_variant  
      break if @mapped_variants.compact.count == @requirement_sets.count  
    end  
  end  
end
```

How do we make our barista
smarter and more personal?

Just relax, one bit at a time

```
def call
  [
    { unique_roaster_constraint: true,  recent_item_constraint: true },
    { unique_roaster_constraint: true,  recent_item_constraint: false },
    { unique_roaster_constraint: false, recent_item_constraint: true },
    { unique_roaster_constraint: false, recent_item_constraint: false },
  ].each do |constraints|
    @mapped_variants = Array.new(@requirement_sets.count)
    map! (**constraints)
    break if @done
  end
end
```

Just relax, one bit at a time

```
def call
  [
    { unique_roaster_constraint: true,  recent_item_constraint: true }, ←
    { unique_roaster_constraint: true,  recent_item_constraint: false },
    { unique_roaster_constraint: false, recent_item_constraint: true },
    { unique_roaster_constraint: false, recent_item_constraint: false },
  ].each do |constraints|
    @mapped_variants = Array.new(@requirement_sets.count)
    map! (**constraints)
    break if @done
  end
end
```

Just relax, one bit at a time

```
def call
  [
    { unique_roaster_constraint: true,  recent_item_constraint: true },
    { unique_roaster_constraint: true,  recent_item_constraint: false }, ←
    { unique_roaster_constraint: false, recent_item_constraint: true },
    { unique_roaster_constraint: false, recent_item_constraint: false },
  ].each do |constraints|
    @mapped_variants = Array.new(@requirement_sets.count)
    map! (**constraints)
    break if @done
  end
end
```

Just relax, one bit at a time

```
def call
  [
    { unique_roaster_constraint: true,  recent_item_constraint: true },
    { unique_roaster_constraint: true,  recent_item_constraint: false },
    { unique_roaster_constraint: false, recent_item_constraint: true }, ←
    { unique_roaster_constraint: false, recent_item_constraint: false },
  ].each do |constraints|
    @mapped_variants = Array.new(@requirement_sets.count)
    map! (**constraints)
    break if @done
  end
end
```

Just relax, one bit at a time

```
def call
  [
    { unique_roaster_constraint: true,  recent_item_constraint: true },
    { unique_roaster_constraint: true,  recent_item_constraint: false },
    { unique_roaster_constraint: false, recent_item_constraint: true },
    { unique_roaster_constraint: false, recent_item_constraint: false }, ←
  ].each do |constraints|
    @mapped_variants = Array.new(@requirement_sets.count)
    map! (**constraints)
    break if @done
  end
end
```

How do we relax anyway?

```
def map!(unique_roaster_constraint:, recent_item_constraint:)
  @products.each do |product|
    @requirement_sets.each_with_index do |requirement_set, index|
      next unless empty_slot?(index)
      next unless requirement_set.matches?(product)
      eligible_variant = product.default_variant
      next unless enough_inventory?(eligible_variant, @mapped_variants)

      next if unique_roaster_constraint && roaster_in_mapped_variants?(eligible_variant)
      next if recent_item_constraint && received_recently?(eligible_variant)

      @mapped_variants[index] = eligible_variant
      if @mapped_variants.compact.count == @requirement_sets.count
        @done = true; break
      end
    end
  end
end
```

How do we relax anyway?

```
def map!(unique_roaster_constraint:, recent_item_constraint:)
  @products.each do |product|
    @requirement_sets.each_with_index do |requirement_set, index|
      next unless empty_slot?(index)
      next unless requirement_set.matches?(product)
      eligible_variant = product.default_variant
      next unless enough_inventory?(eligible_variant, @mapped_variants)

      next if unique_roaster_constraint && roaster_in_mapped_variants?(eligible_variant)
      next if recent_item_constraint && received_recently?(eligible_variant)

      @mapped_variants[index] = eligible_variant
      if @mapped_variants.compact.count == @requirement_sets.count
        @done = true; break
      end
    end
  end
end
```

How do we relax anyway?

```
def map!(unique_roaster_constraint:, recent_item_constraint:)
  @products.each do |product|
    @requirement_sets.each_with_index do |requirement_set, index|
      next unless empty_slot?(index)
      next unless requirement_set.matches?(product)
      eligible_variant = product.default_variant
      next unless enough_inventory?(eligible_variant, @mapped_variants)

      next if unique_roaster_constraint && roaster_in_mapped_variants?(eligible_variant)
      next if recent_item_constraint && received_recently?(eligible_variant)

      @mapped_variants[index] = eligible_variant
      if @mapped_variants.compact.count == @requirement_sets.count
        @done = true; break
      end
    end
  end
end
```

The invisible barista was taking
a hold

The harder upgrade

The system got better while “the answers” got worse.

The question	The answer before	The answer after
Why did slot 3 get this coffee?	“At that time, it was the first in stock product matching roast level, caffeine, etc...”	“Hmm. Maybe the customer received the other coffees recently, maybe we had a temporary inventory issue. Just maybe.”

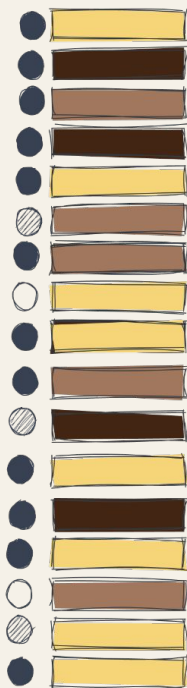
The team (gradually) accepted the tradeoff.

A more elegant way

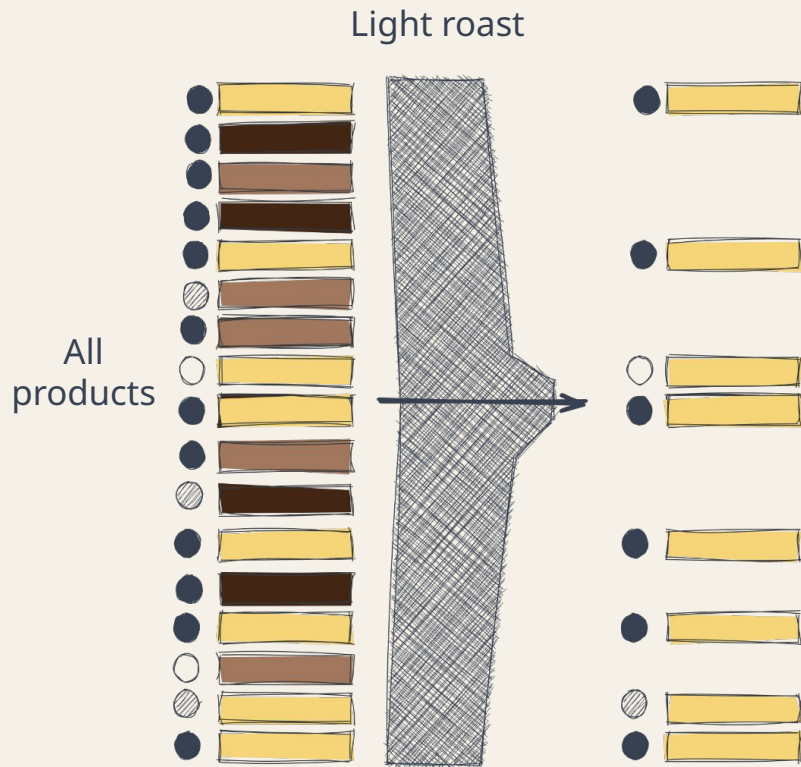
You shall not pass!

You shall not pass!

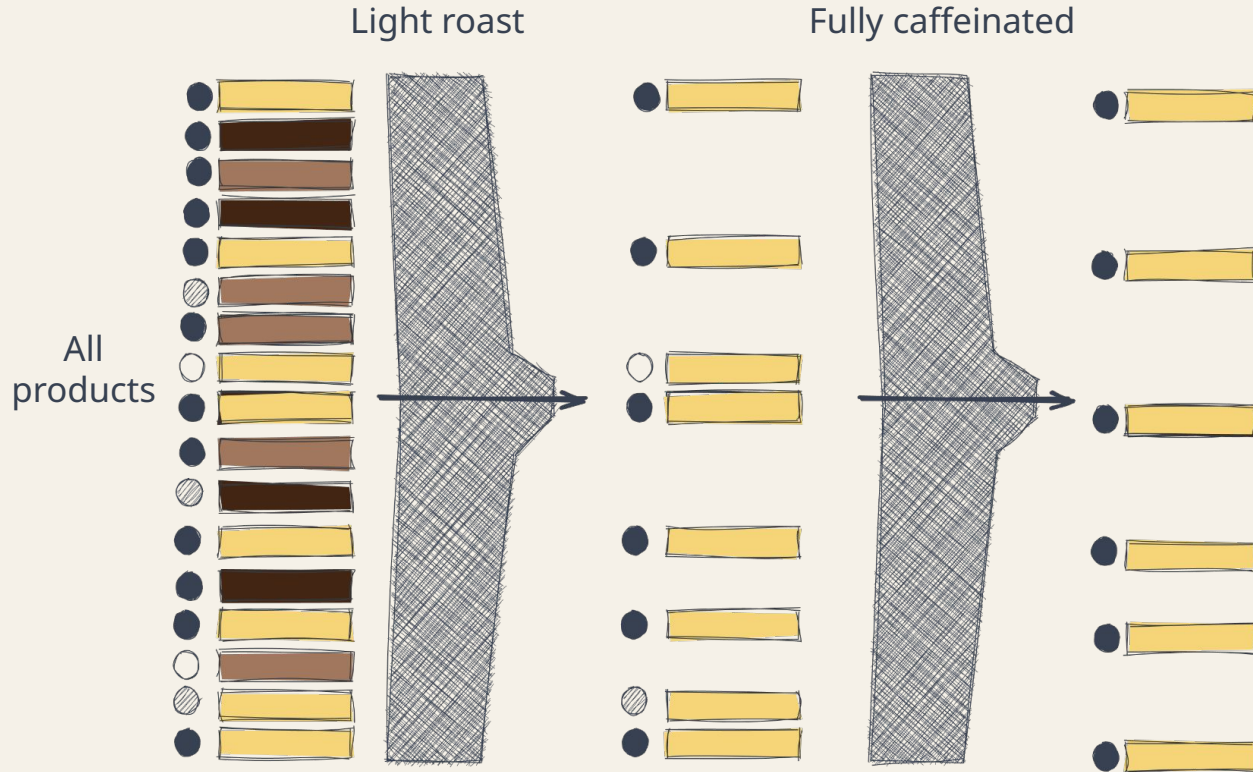
All
products



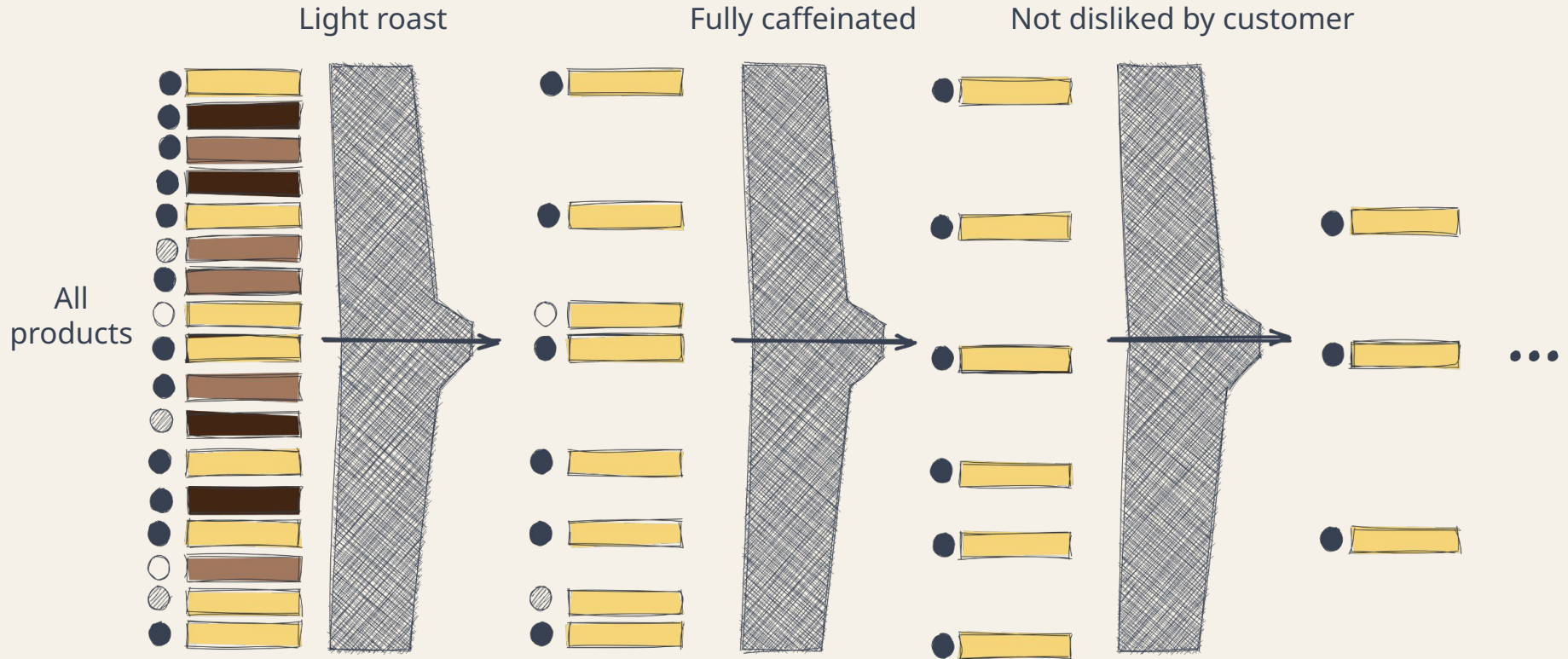
You shall not pass!



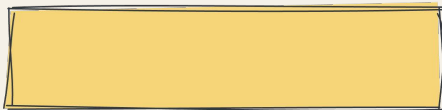
You shall not pass!



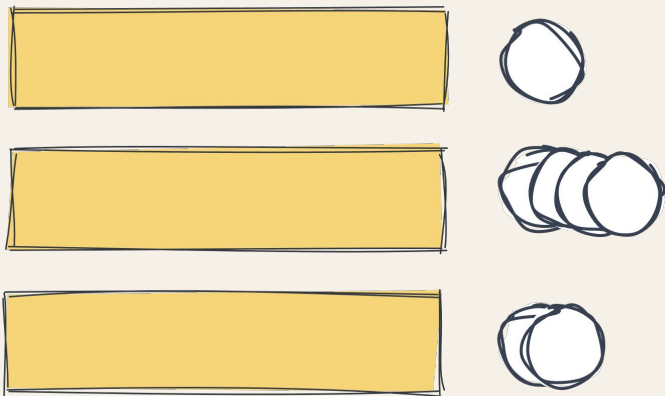
You shall not pass!



The Sorting Hat Has No Opinions



The Sorting Hat Has No Opinions



The Sorting Hat Has No Opinions



One does not simply count conditions

Limitations of counting conditions as votes:

- Every condition gets one vote, but not all conditions are equal
- “Different roaster” and “coffee never seen before” aren’t the same magnitude
- A product that wins 3 “minor” conditions beats a product winning only 2 “critical” ones

To filter or not to filter, that's a lot of booleans

```
class WeightedMapper
  def call
    @mappings = []

    @requirement_sets.each do |requirement_set|
      eligible_products = @products.select do |product|
        FILTERS.all? do |filter|
          filter.new(product:, requirement_set:, mappings:).eligible?
        end
      end
    end
    # ...
  end
end
```

Wait! Or weigh?

Everybody gets a weight!

```
# ...
results = eligible_products.map do |product|
  weight = 1
  amplifications = [ { name: 'base', magnitude: 1 } ]
  PARAMETERS.each do |param|
    amplifier = param[:class].new(param[:coeff], product, ...)
    magnitude = amplifier.magnitude
    weight *= magnitude
    amplifications << { name: amplifier.name, magnitude: }
  end
  Result.new(product:, weight:, amplifications:)
end
# ...
```

Everybody gets a weight!

```
# ...
results = eligible_products.map do |product|
  weight = 1
  amplifications = [ { name: 'base', magnitude: 1 } ]
  PARAMETERS.each do |param|
    amplifier = param[:class].new(param[:coeff], product, ...)
    magnitude = amplifier.magnitude
    weight *= magnitude
    amplifications << { name: amplifier.name, magnitude: }
  end
  Result.new(product:, weight:, amplifications:)
end
# ...
```

Everybody gets a weight!

```
# ...
results = eligible_products.map do |product|
  weight = 1
  amplifications = [ { name: 'base', magnitude: 1 } ]
  PARAMETERS.each do |param|
    amplifier = param[:class].new(param[:coeff], product, ...)
    magnitude = amplifier.magnitude
    weight *= magnitude
    amplifications << { name: amplifier.name, magnitude: }
  end
  Result.new(product:, weight:, amplifications:)
end
# ...
```

Everybody gets a weight!

```
# ...
results = eligible_products.map do |product|
  weight = 1
  amplifications = [ { name: 'base', magnitude: 1 } ]
  PARAMETERS.each do |param|
    amplifier = param[:class].new(param[:coeff], product, ...)
    magnitude = amplifier.magnitude
    weight *= magnitude
    amplifications << { name: amplifier.name, magnitude: }
  end
  Result.new(product:, weight:, amplifications:)
end
# ...
```

Everybody gets a weight!

```
# ...
results = eligible_products.map do |product|
  weight = 1
  amplifications = [ { name: 'base', magnitude: 1 } ]
  PARAMETERS.each do |param|
    amplifier = param[:class].new(param[:coeff], product, ...)
    magnitude = amplifier.magnitude
    weight *= magnitude
    amplifications << { name: amplifier.name, magnitude: }
  end
  Result.new(product:, weight:, amplifications:)
end
# ...
```

Winner takes the spot

```
# ...
sorted_results = results.sort_by { _1.weight * -1 }
winner = sorted_results.shift
raise UnmappableRequirement unless winner
winner.metadata[:top_competing] = sorted_results.first(3).map {
  { sku: _1.variant.sku, weight: _1.weight }
}
@mappings << winner
end
end
end
```

What's in an amplifier anyway?

A node of few words

```
class RoasterNotInBox < Weight
  def amplification
    if roaster_in_current_mappings?
      1
    else
      coefficient
    end
  end

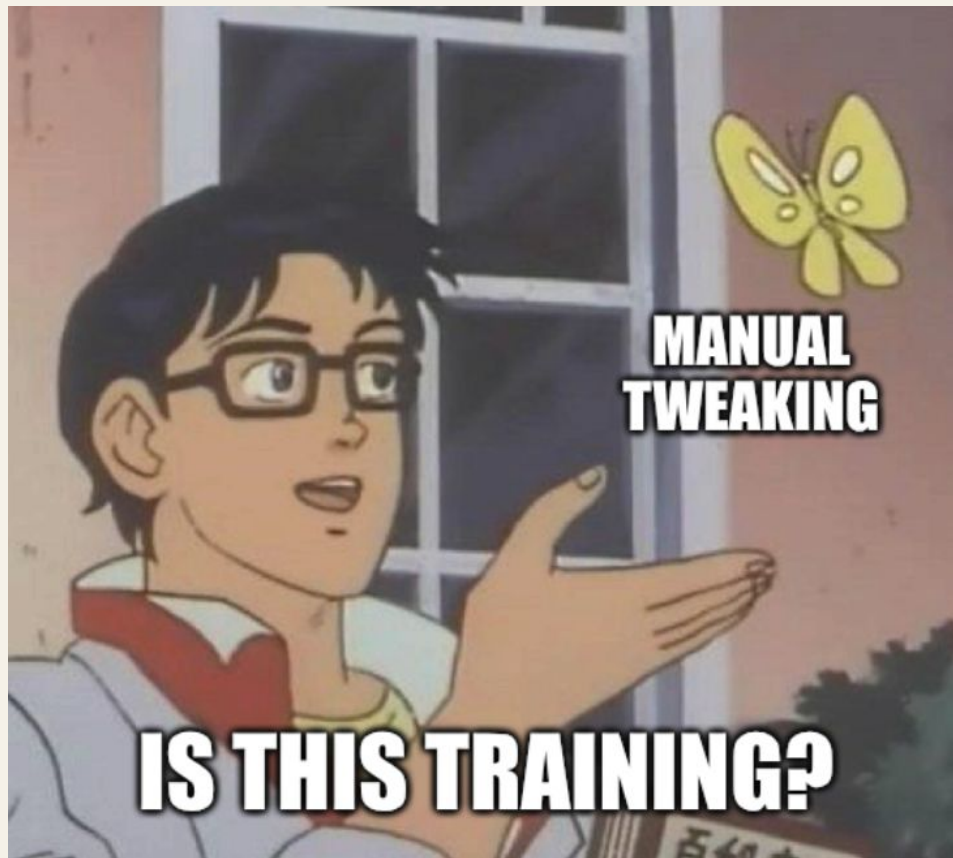
  private
  ...
end
```

Compound interest, compound disappointment

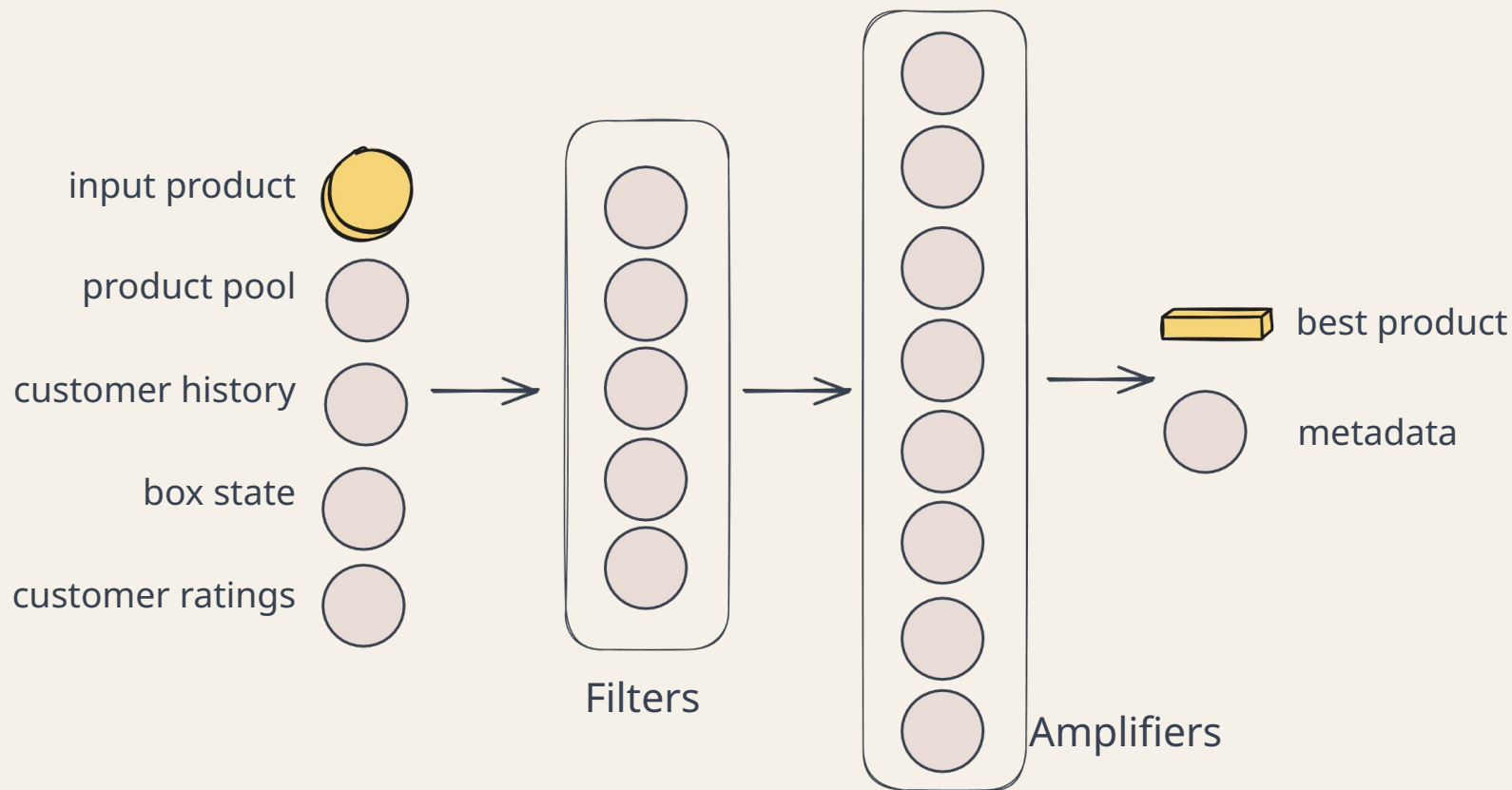
```
class BoxVariety < Weight
  def amplification
    if times_this_product_is_in_current_mappings.zero?
      coefficient
    else
      1 / times_this_product_is_in_current_mappings.to_f
    end
  end

  private
  ...
end
```

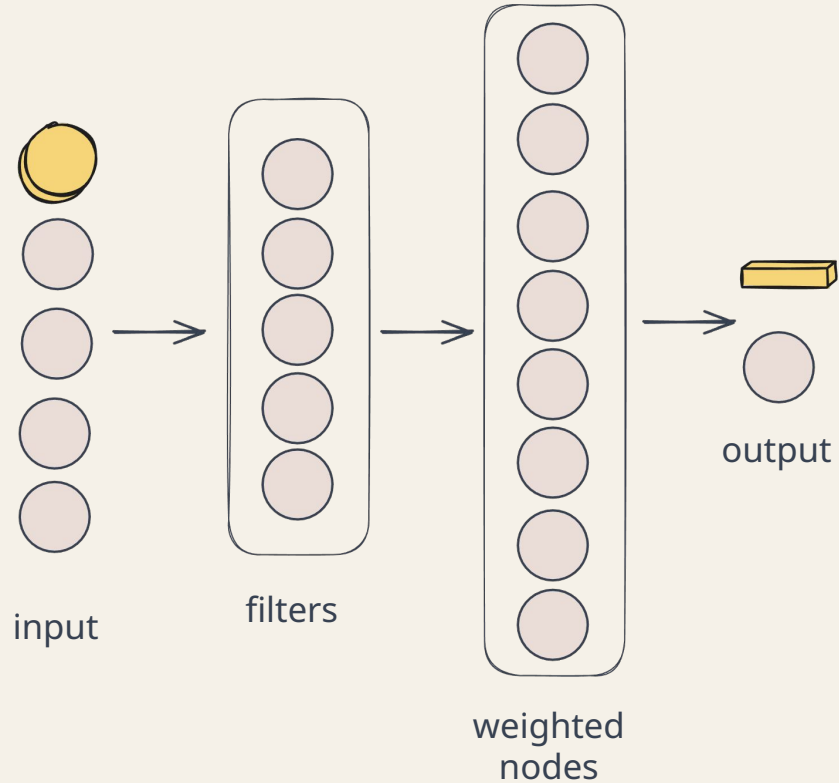
Tweak. Deploy. Observe. Repeat.



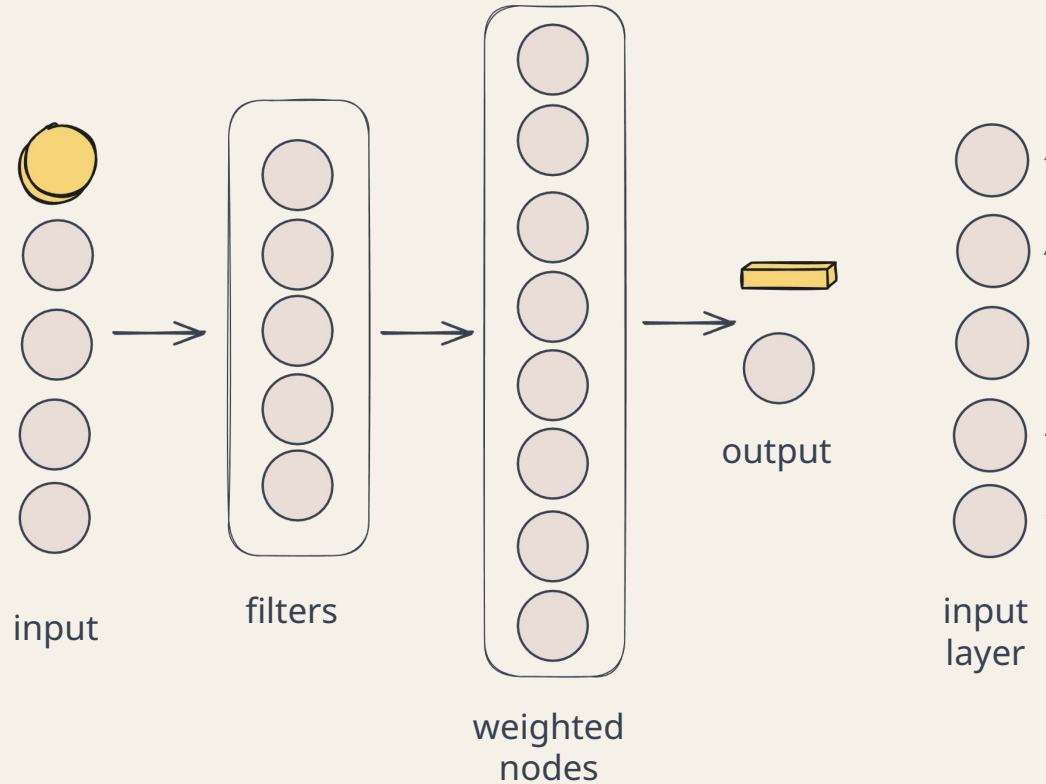
What if I told you...



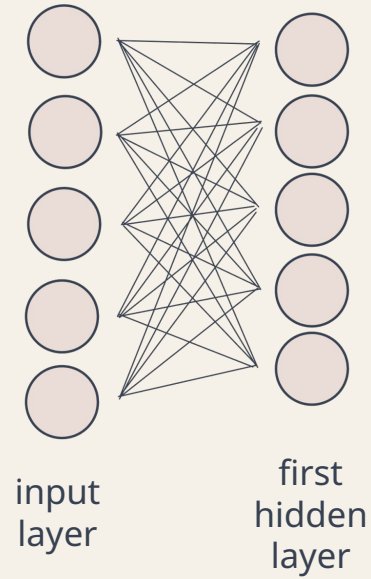
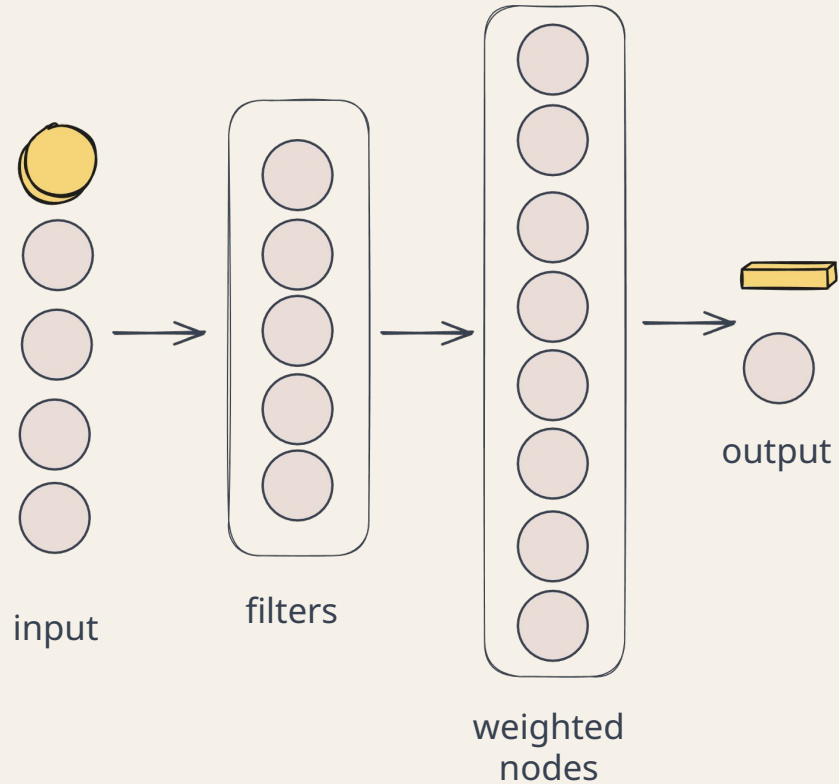
... it already has a name



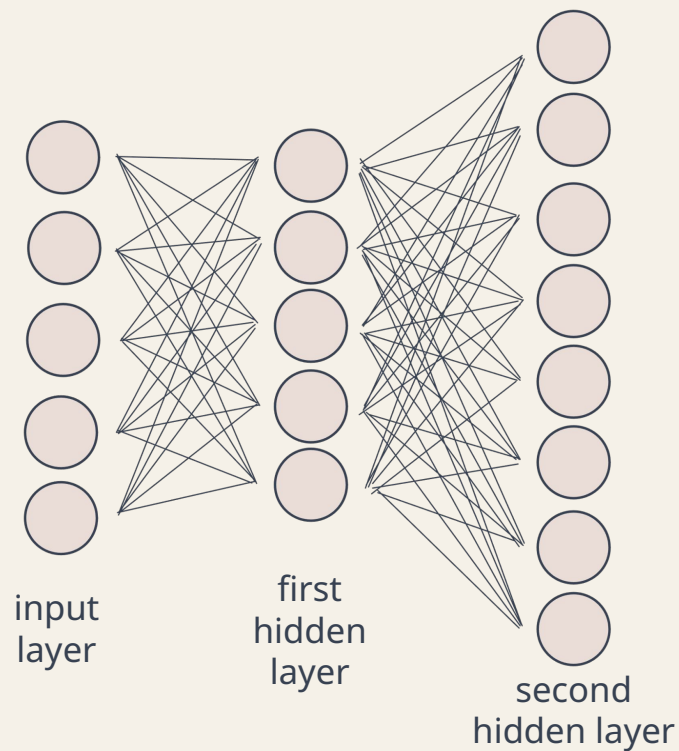
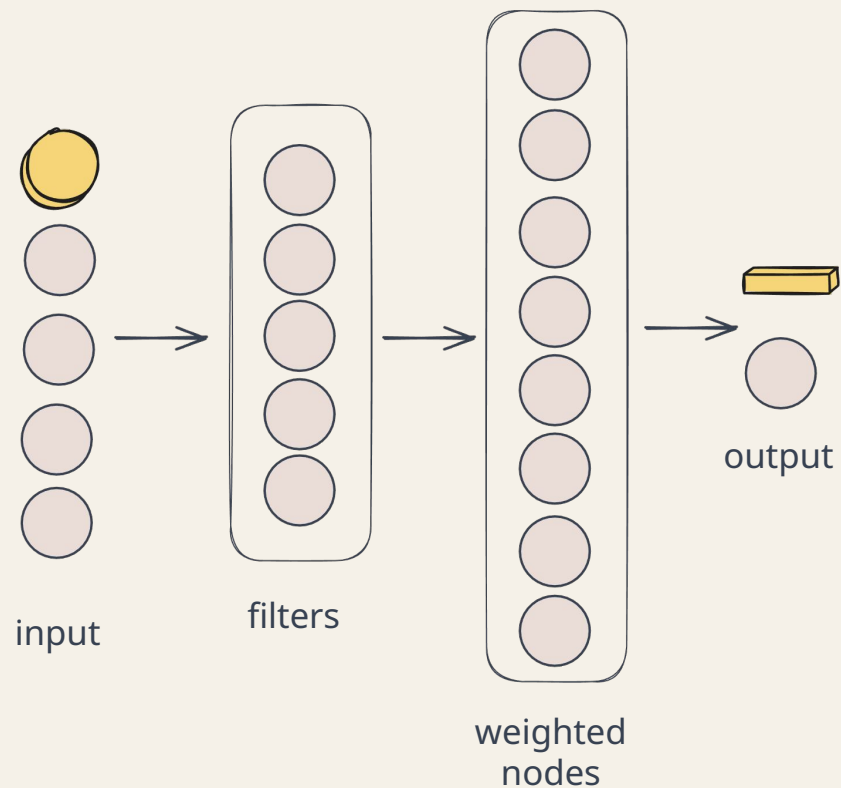
... it already has a name



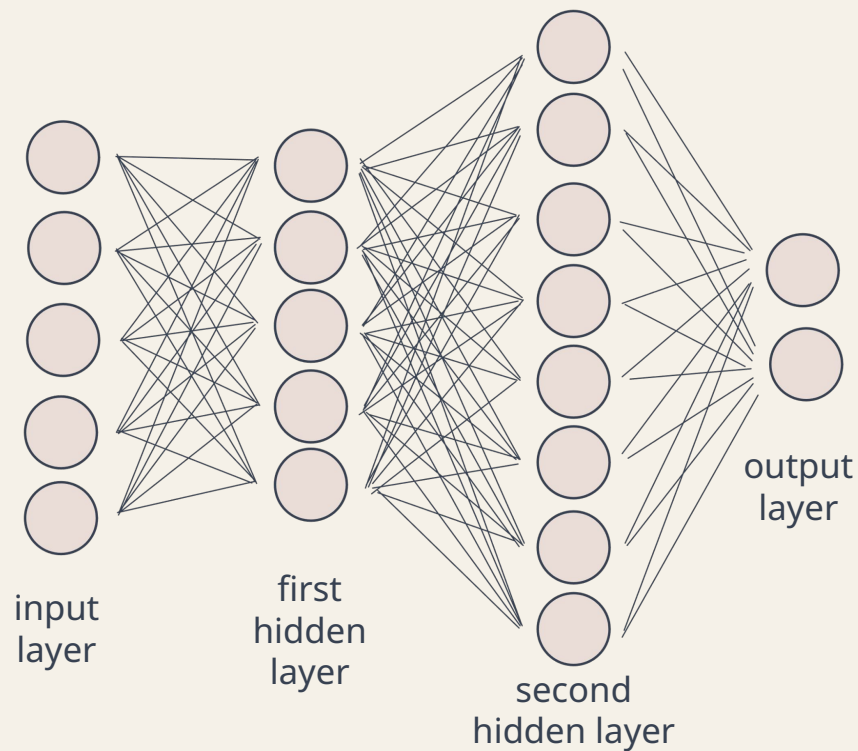
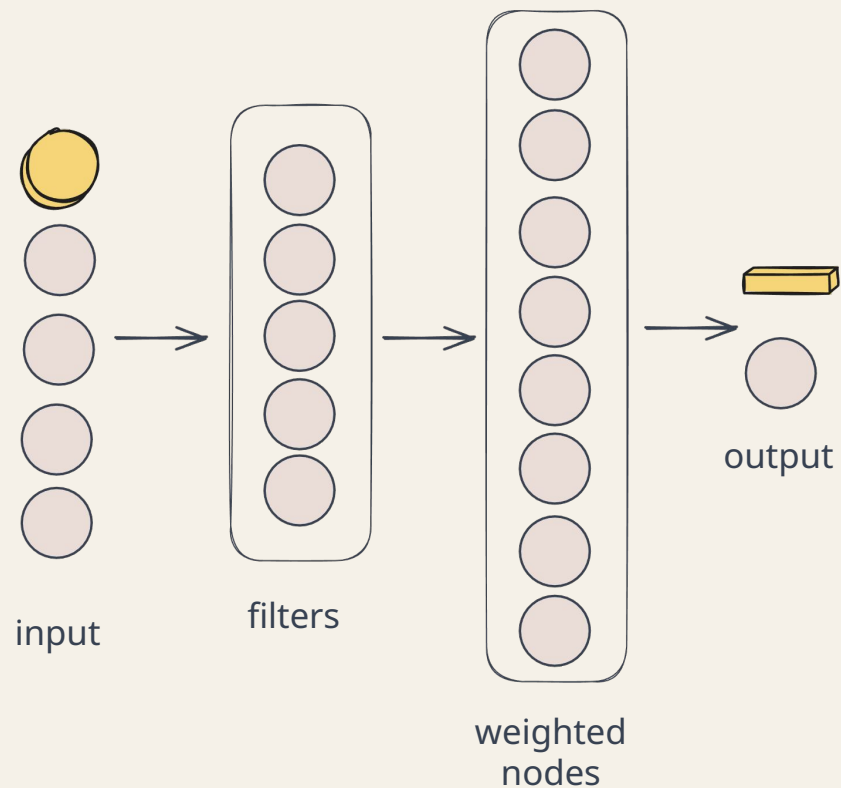
... it already has a name



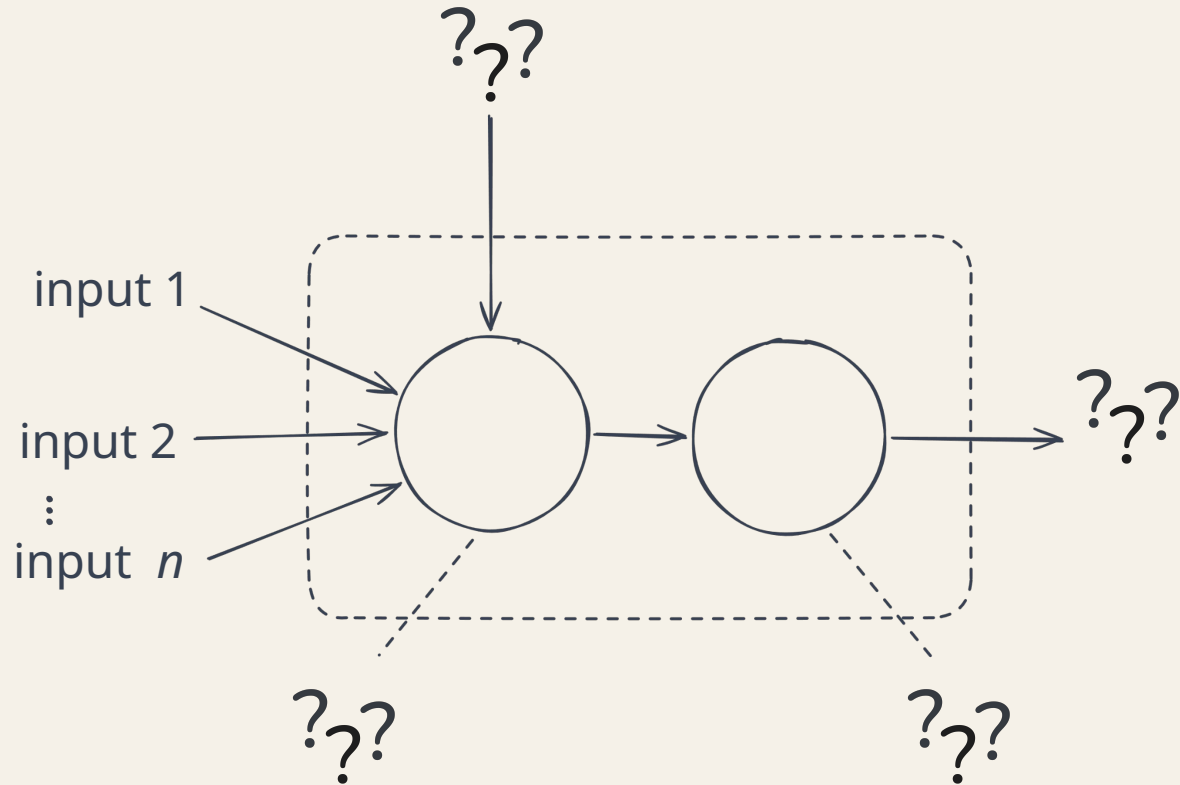
... it already has a name



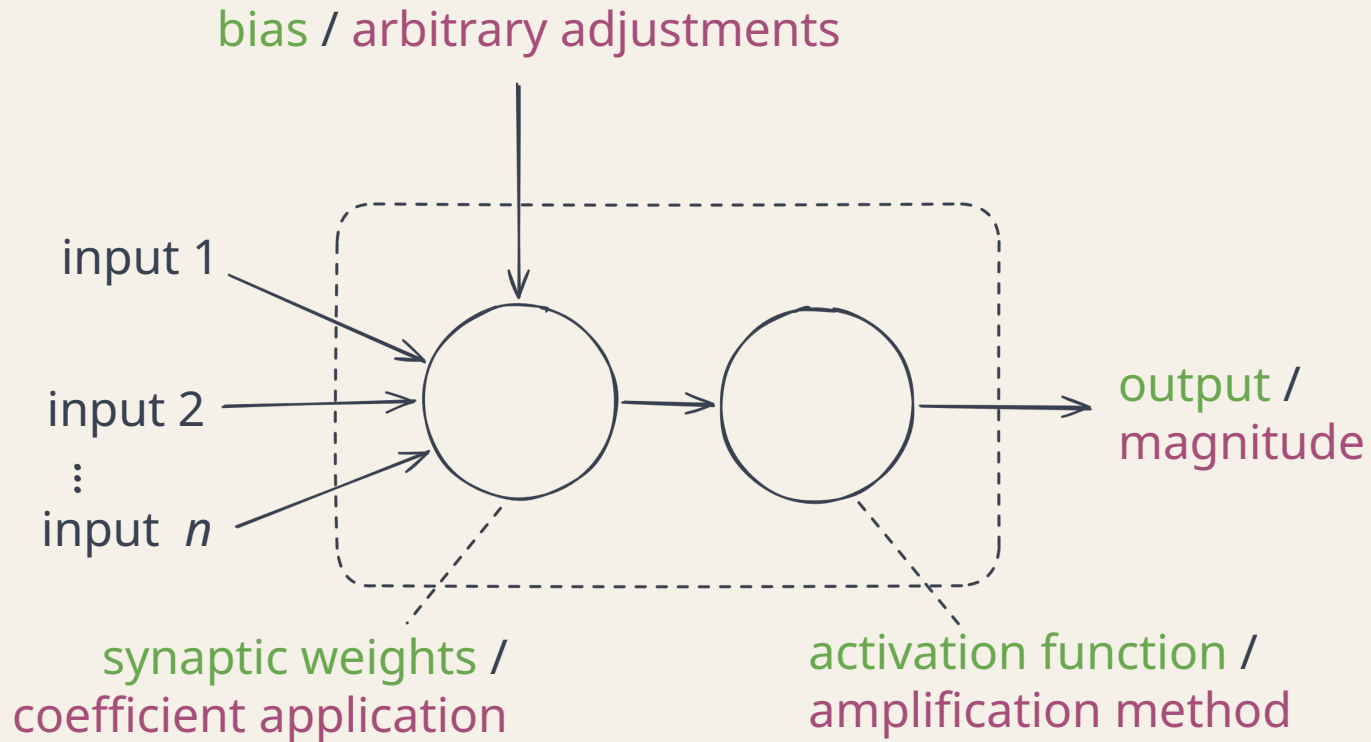
... it already has a name



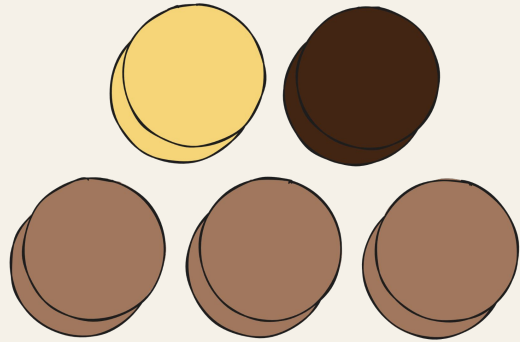
What's in a node, or is it a neuron?



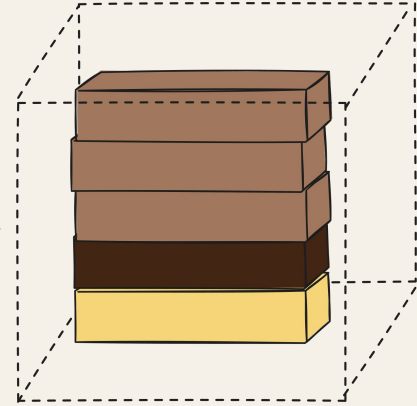
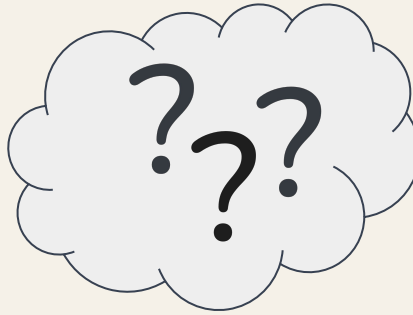
What's in a node, or is it a neuron?



So what has this become?

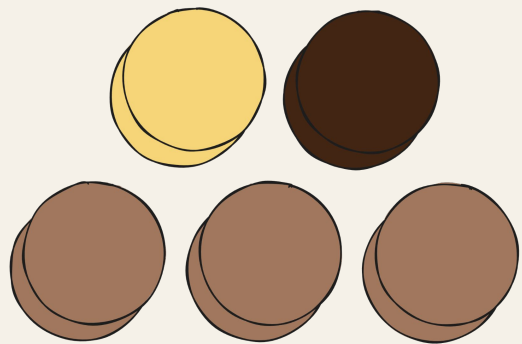


What the customer buys:
5 virtual products

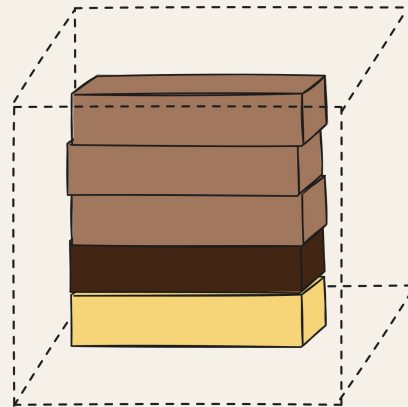


What the warehouse ships:
5 tangible SKUs

So what has this become?



What the customer buys:
5 virtual products



What the warehouse ships:
5 tangible SKUs

This is how NOT to build a neural
network

A neural network is a weighted decision graph

Inputs flow through layers of nodes

Each makes a local computation

Passes a number forward

The shape of it is what we just built in Ruby

The right tool for the problem

The bottleneck was never the Ruby, it was the Shopify API.

Manual tuning = explainable decisions = happy stakeholders.

The problem was still small enough that a few humans could actually think it through.

Building without a blueprint

No blueprint.

No framework.

Almost no idea about what we were building.

(It won't always work out)

When to Use This vs. Graduate to ML

This pattern works when...	Graduate to proper ML when...
Problem is small enough to reason about	Data volume makes manual tuning impractical
Rules are clear enough to name	Relationships between inputs are too complex to describe
Interpretability matters more than last-% accuracy	System needs to discover patterns you haven't thought of
You can enumerate what "good" looks like	You're tuning 50 coefficients by hand and your head melts

You may have already built a neural network, and undersold it as a service object.

Accidentally building a Neural Network

A Ruby product recommendation journey

Nicolò Rebughini