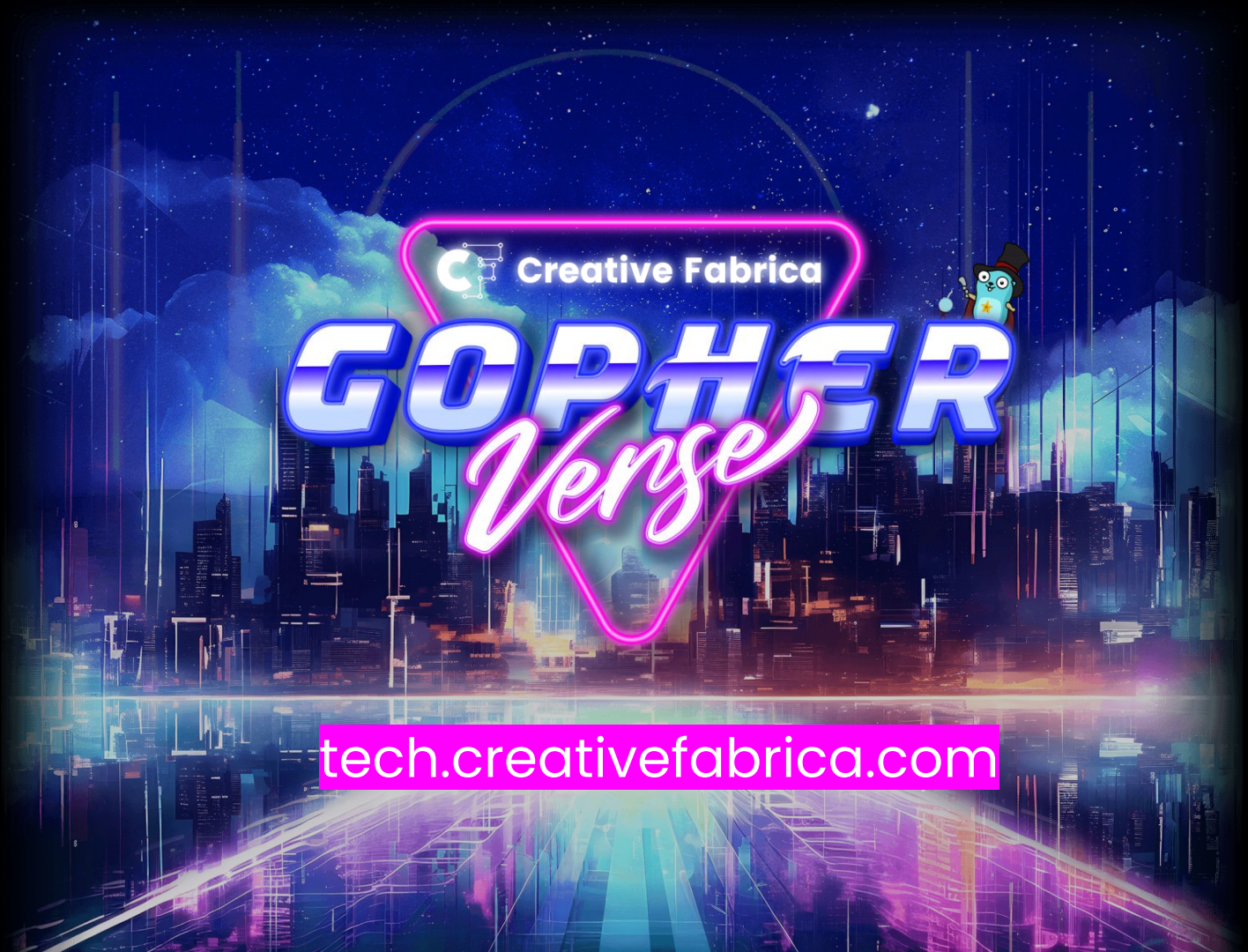
Creative Fabrica

How we built Gopher-Verse

Herlon Aguiar

Staff Software Engineer

E-Mail
herlon@creativefabrica.com


Creative Fabrica
GOPHER
Verse
tech.creativefabrica.com

What kind of gopher do you want to generate?

happy go blue gopher in the forest

Generate ✦

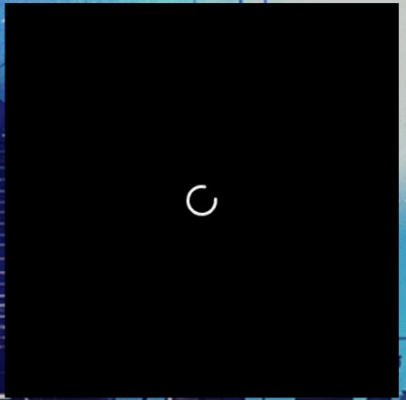
Latest Generations

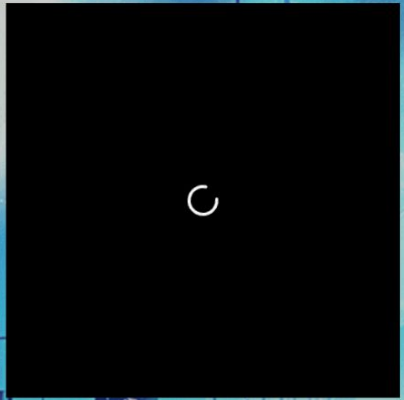
go blue gopher in the style of starry night

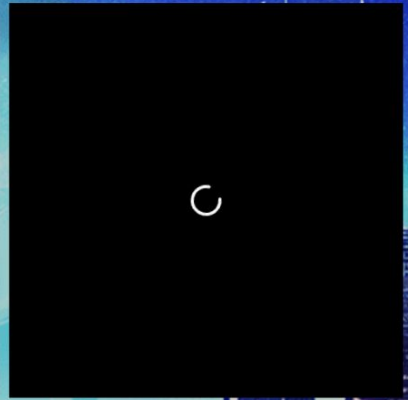
Generate

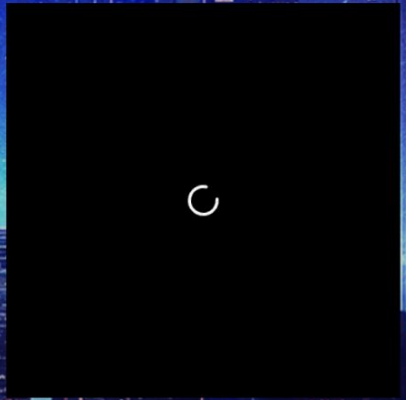
Generating your gophers

You are the next in the queue.









Go Blue Gopher in the style of Starry Night.

New prompt

We're hiring!

Check out our open positions.

Go Blue Gopher in the style of Starry Night.

New prompt

tech.creativefabrica.com

We're hiring!
Check out our open positions.

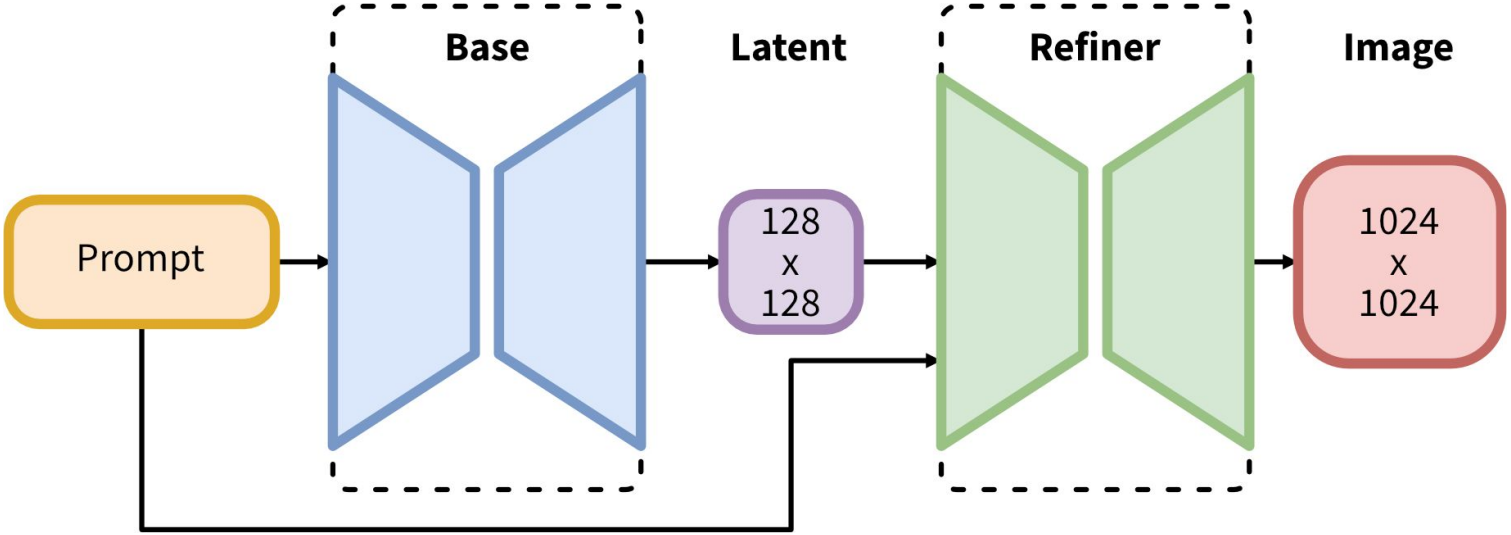
What’s inside the box?

What’s in the box?

Stable Diffusion XL

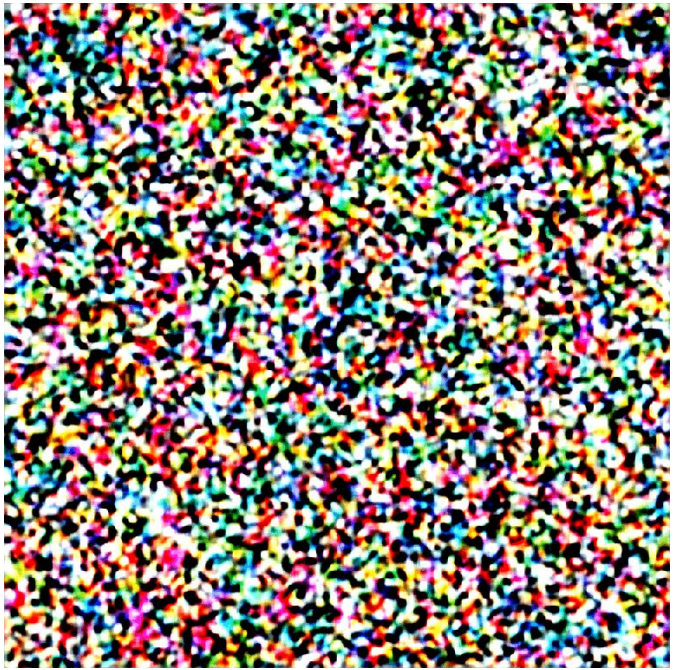
stability.ai

Stable Diffusion XL



Stable Diffusion XL

Stable Diffusion XL



Stable Diffusion XL - License

CreativeML Open RAIL++-M License by Stability AI

Permissions: You can use, modify, distribute, and create derivatives of the model.

Restrictions: You cannot use the model for:

- Illegal activities
- Exploiting minors
- Generating false information to harm others
- Creating personal identifiable information that could harm individuals
- Defaming or harassing others
- Fully automated decision-making that affects legal rights
- Discriminating against individuals or groups
- Exploiting vulnerabilities of specific groups
- Providing medical advice
- Generating information for law enforcement, immigration, or justice system processes

Output ownership: The license makes no claims on the output you generate, but you're accountable for how you use it.
Liability: The model is provided "AS IS" without warranties, and the licensor isn't liable for damages.

Redistribution requirements: If you distribute the model or derivatives, you must include these same use restrictions.

Stable Diffusion XL - License

The CreativeML Open RAIL++-M License grants users rights to:

use, **modify**, and **distribute** AI models while **prohibiting specific harmful applications** and requiring redistributors to maintain the same responsible use restrictions.

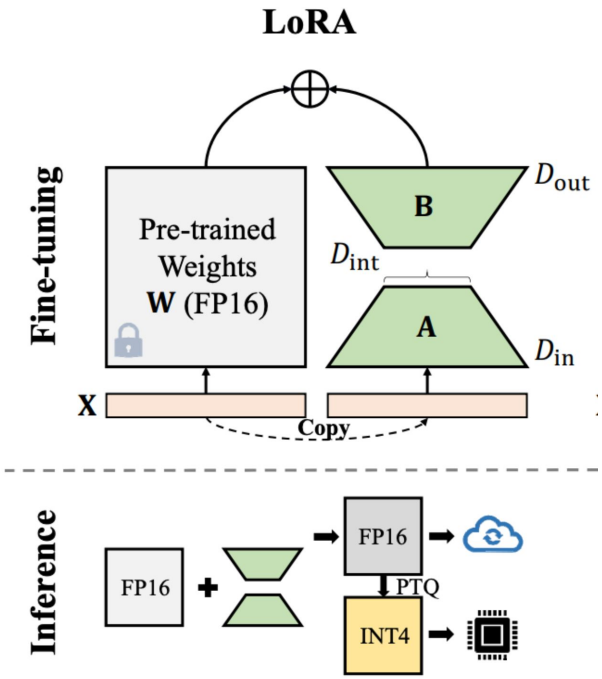
What’s in the box?

LoRA: Low-Rank Adaptation of Large Language Models

LoRa is a technique that makes it easier to customize large AI models without massive computing resources:

1. Instead of retraining the entire model (billions of parameters), LoRa adds small "adapter" matrices to key layers.
2. These adapters create shortcuts that modify the model's behavior while keeping the original weights frozen.
3. **It’s like adding small steering wheels to a large ship** - the main engine stays the same, but you can change direction easily.
4. Much less memory required, trains faster, and lets you switch between different adaptations for different tasks.

LoRA: Low-Rank Adaptation of Large Language Models



Credits: https://cameronrwolfe.substack.com/p/easily-train-a-specialized-llm-peft

What’s in the box?

Trigger word

“Go blue gopher”

What’s in the box?

Trigger word

“**Go blue gopher** as a bodybuilder”

What’s in the box?

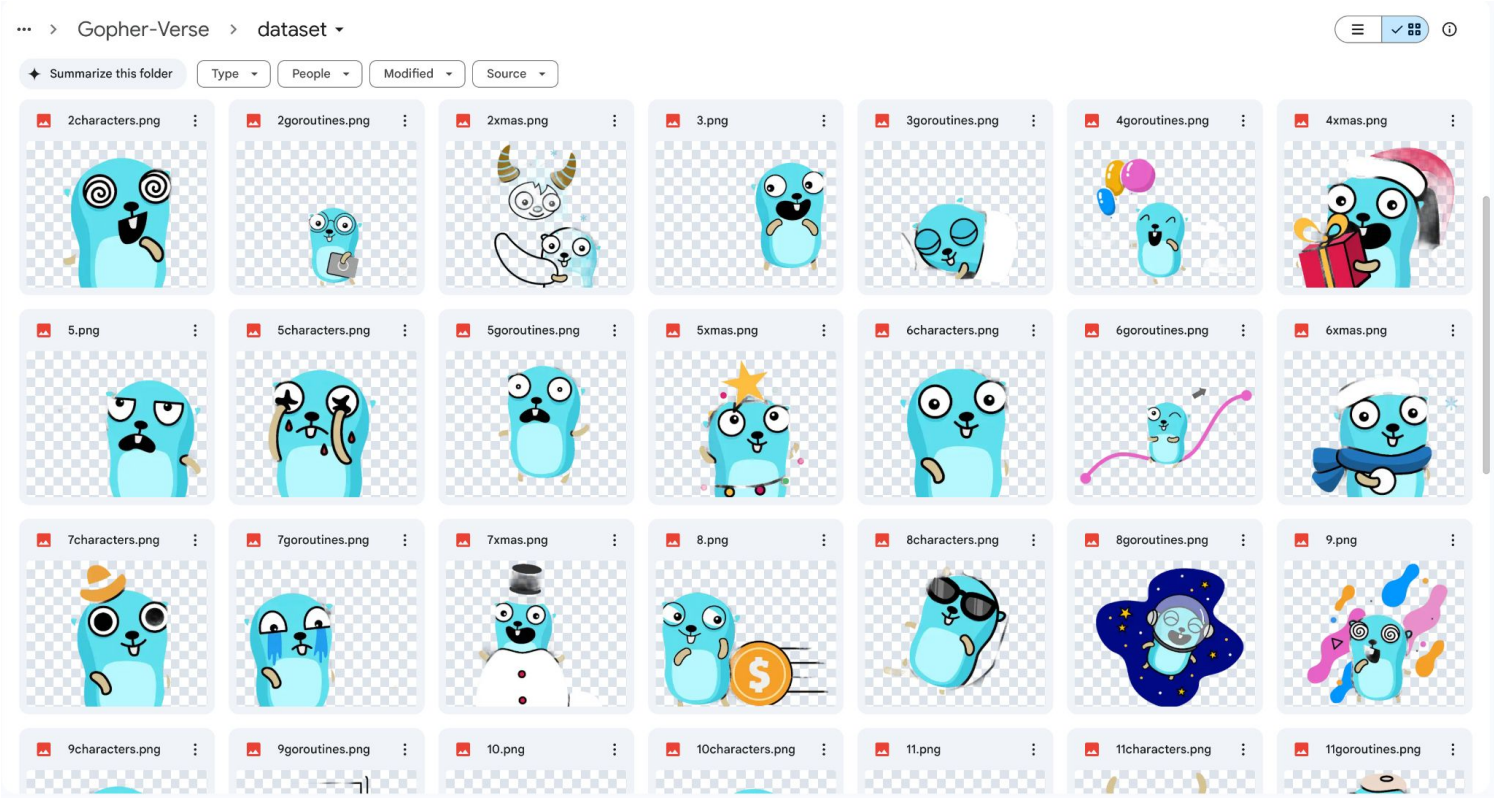
Trigger word

“**Go blue gopher** as a bodybuilder”

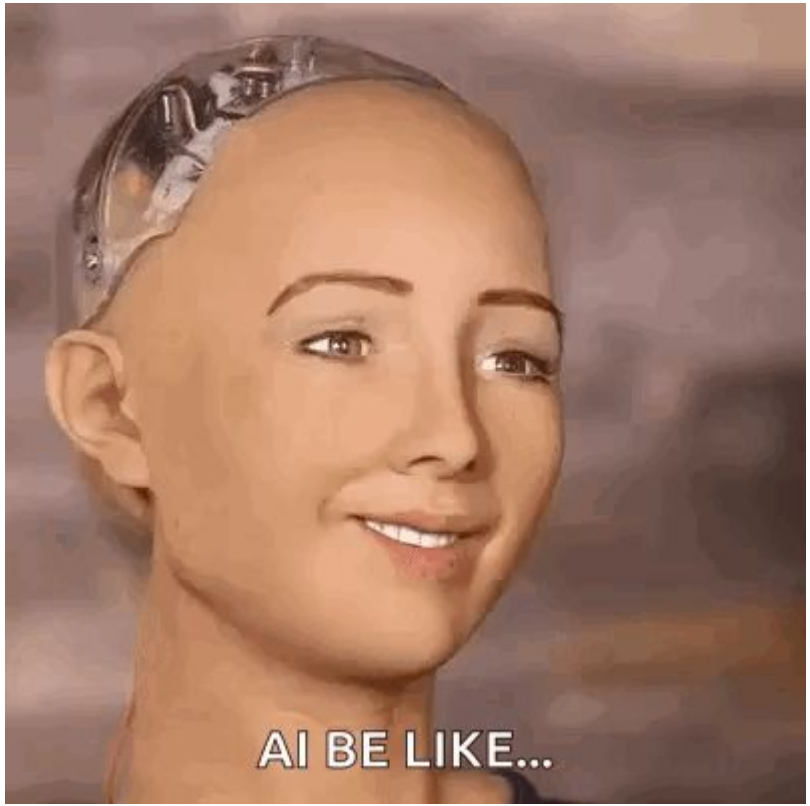
Training Process

Yeah, the boring part

1) Collect images

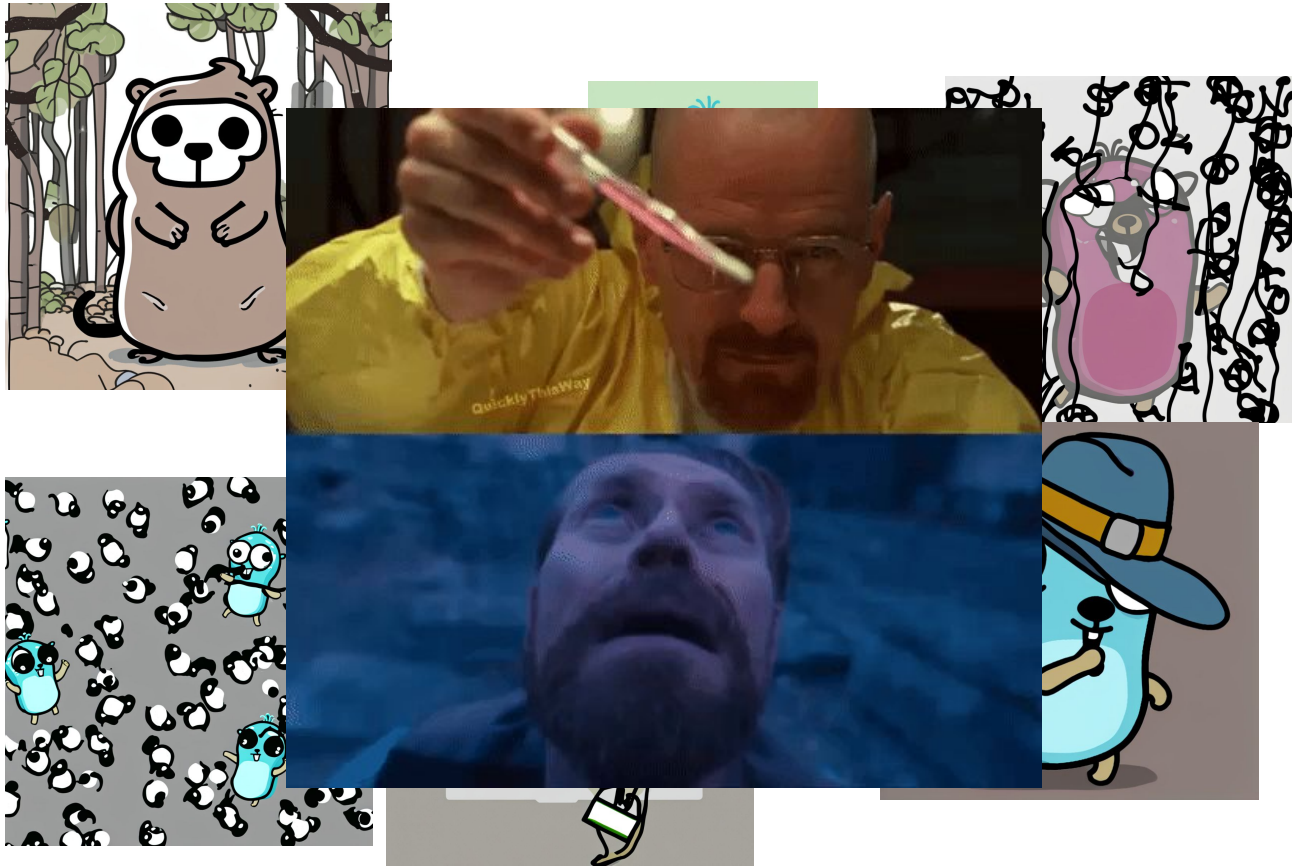


2) Let AI figure out

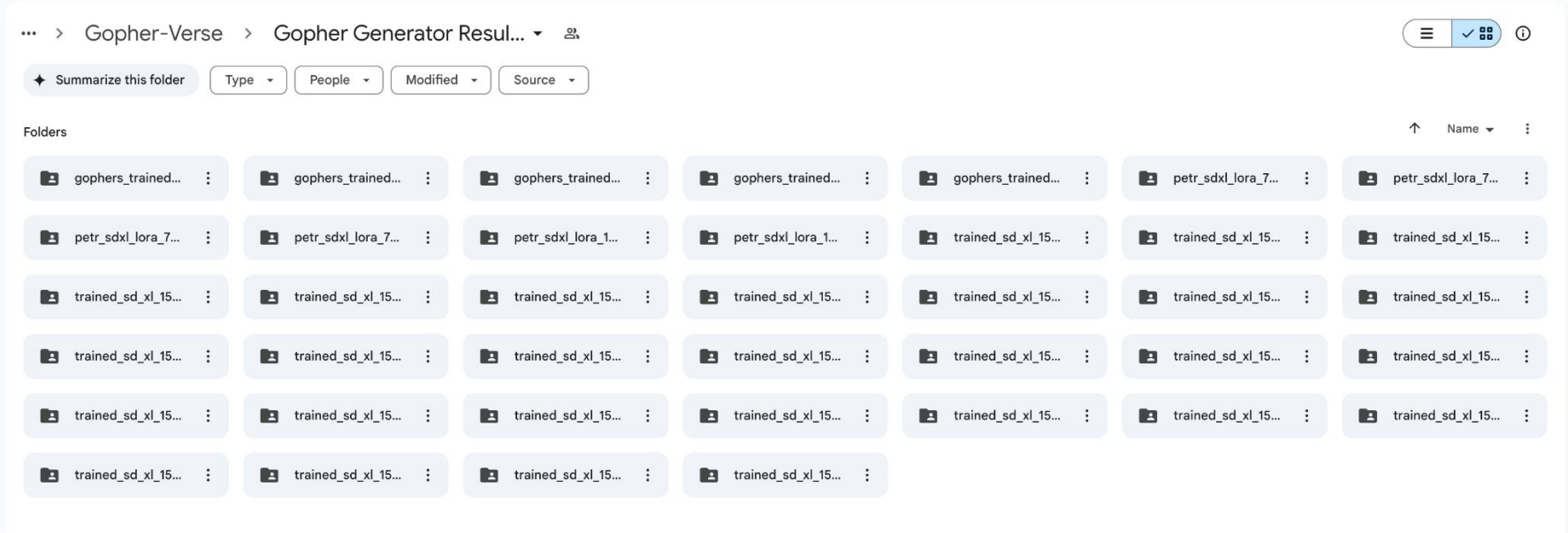


2) Let AI figure out

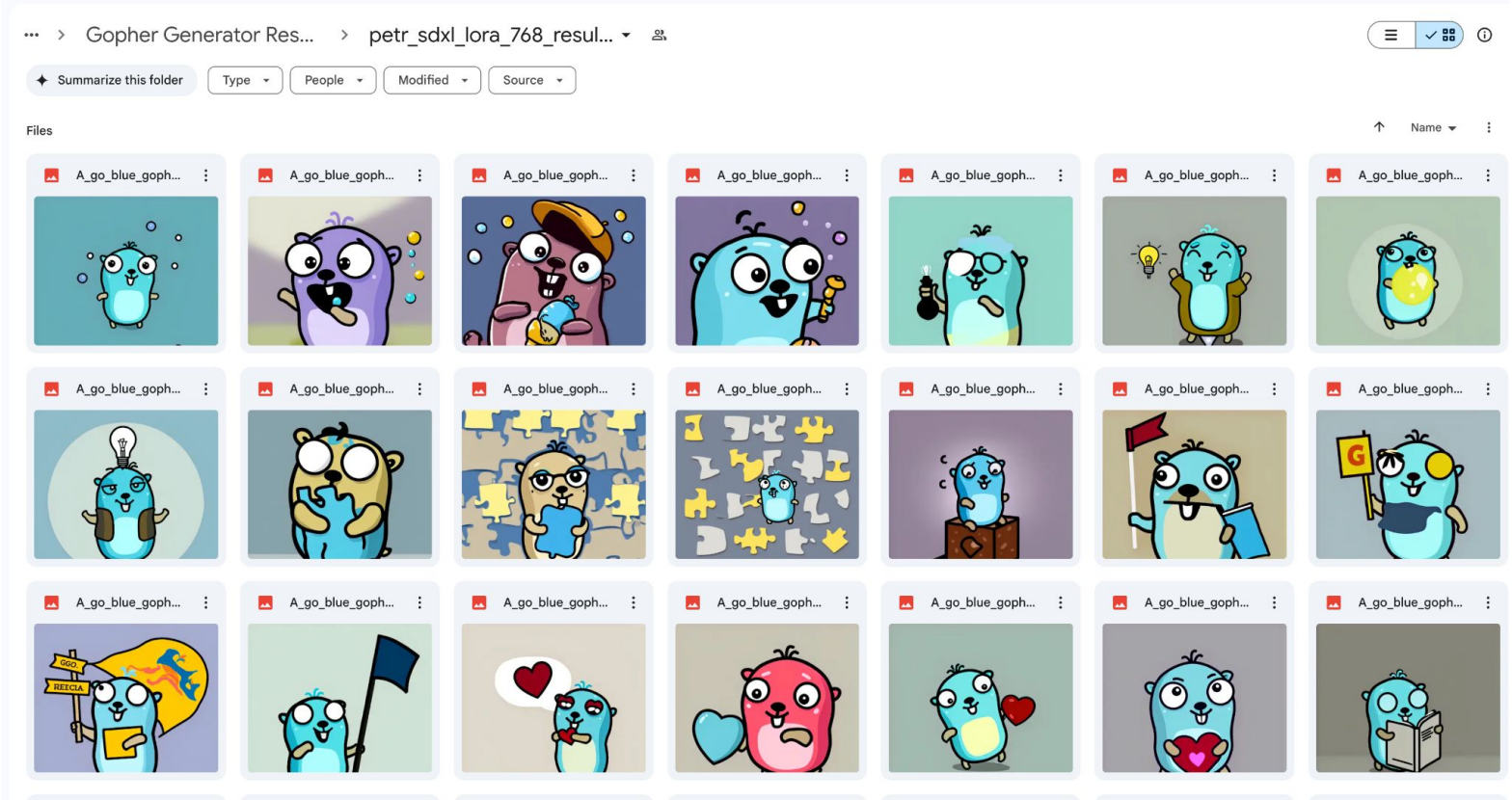
2) Let AI figure out



3) Train, analyze, improve, repeat



2) 3) Train, analyze, improve, repeat



Your project deserve a nice gopher

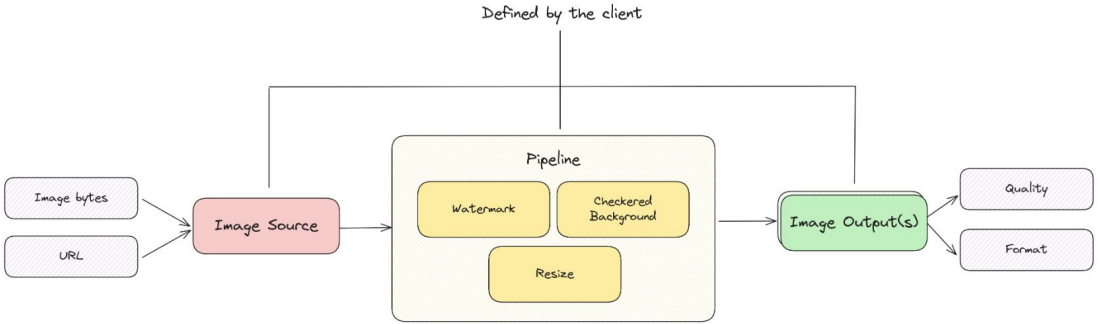
IMP (Image Manipulation Pipeline)

This is a Lambda container that provides image manipulation services. It is written in Go and uses the libvips behind the scenes.

Its main purpose is to provide a simple and efficient way to manipulate images using the power of libvips and the simplicity of calling a Lambda function.

Why there are images commited in the repo? We want to make sure any change we make don't screw the current pipelines and when you make PRs everyone can see what changed in the images and it's easier to spot unexpected changes.

> If you want to run this service locally you need to install **libvips**. You can find the installation instructions here.

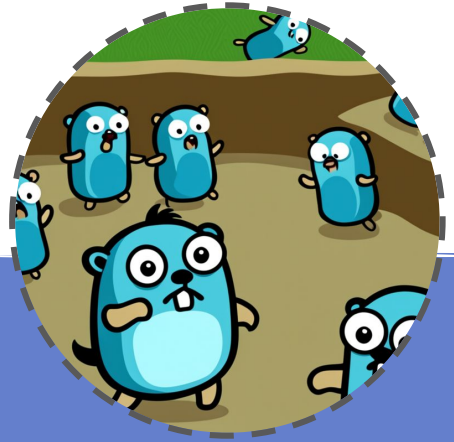


You can print!

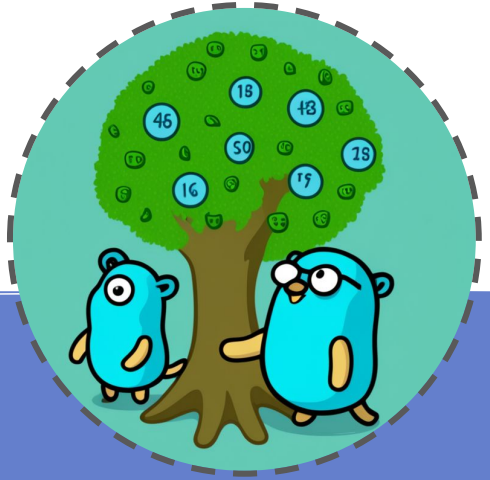
Our printer has limited stock available.

RUN! 🏃

Limitations 😭



Groups



Too technical

"Reversing a binary tree"

Color

"Go Blue Gopher as Pink"

Let’s talk about Go

Of course we have to!

Prompt Filtering

"Gopher in the forest"

↓

ChatGPT

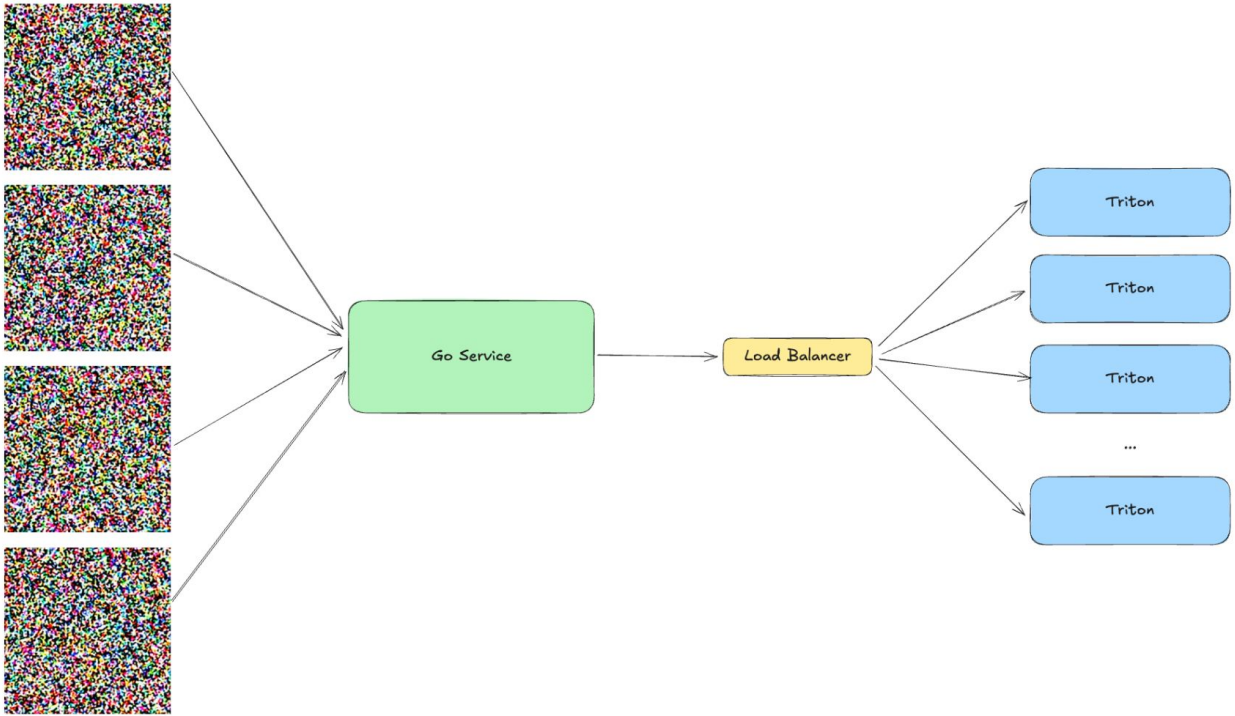
↓

"Go blue gopher in the forest, exploring its surroundings and searching for adventure."

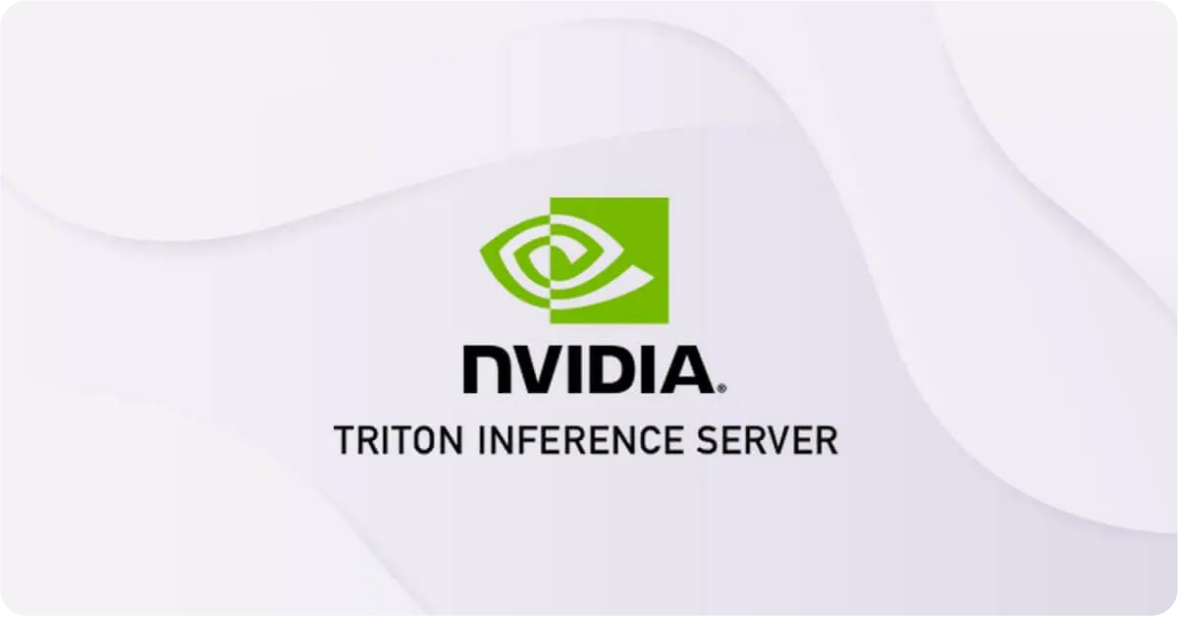
Prompt Filtering

```go
func (r *Resolver) gopherizePrompt(ctx context.Context, prompt string) (string, error) {
  resp, err := r.OpenAIClient.CreateChatCompletion(
    ctx,
    openai.ChatCompletionRequest{
      Model: openai.GPT4Turbo0125,
      Messages: []openai.ChatCompletionMessage{
        {
          Role:    openai.ChatMessageRoleSystem,
          Content: `You're an assistant that helps me gopherize my prompt, you should be short and to the point.`,
        },
        {
          Role:    openai.ChatMessageRoleUser,
          Content: fmt.Sprintf(GopherizerPrompt, prompt),
        },
      },
      Temperature: 0.8,
    },
  )
  if err != nil {
    r.Logger.Error("error creating chat completion", zap.Error(err))

    return prompt, err
  }
```

Image Generation



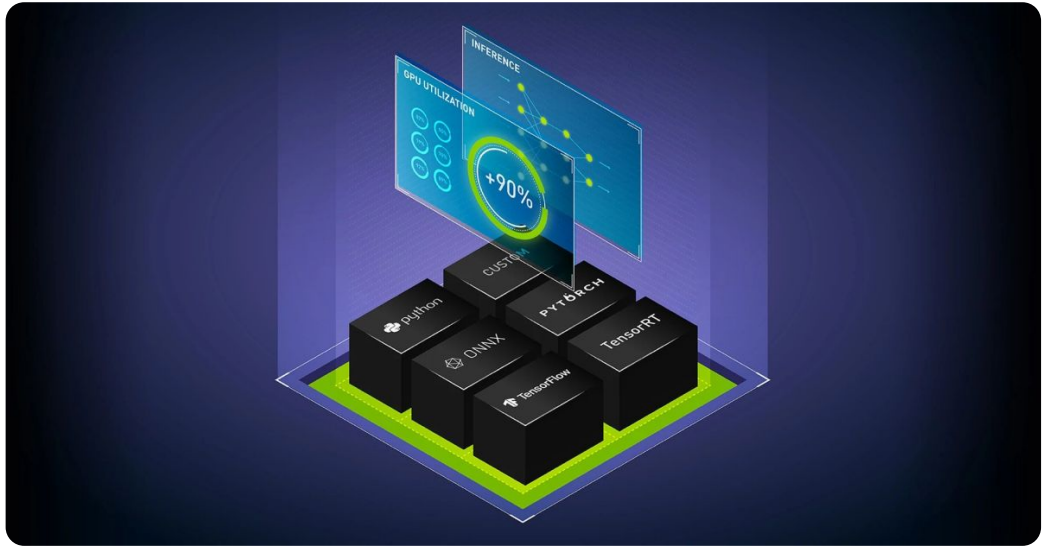
Triton



Credits: https://10clouds.com/blog/a-i/nvidia-triton-production-ready-model-deployment/

Triton

NVIDIA Triton™ Inference Server, part of the **NVIDIA AI** platform and available with NVIDIA AI Enterprise, is **open-source** software that standardizes AI model deployment and execution across every workload.



Triton

Interesting...

Triton

idk, python stuff...

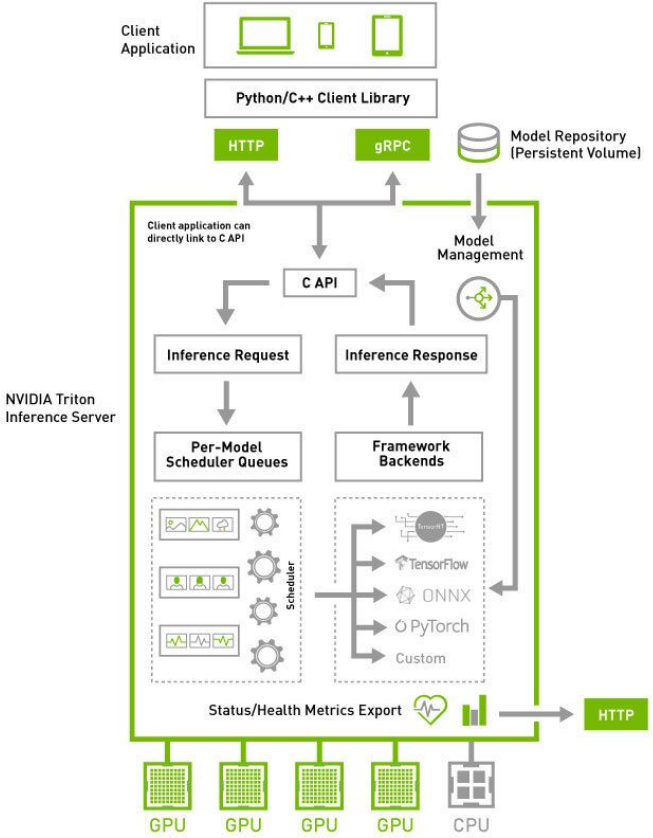
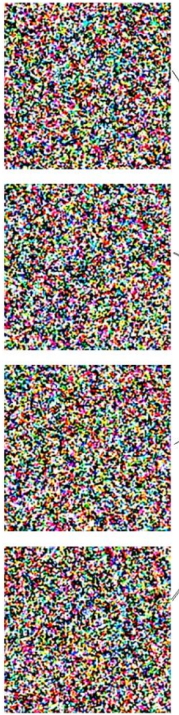


Image Generation



Dive Deep*
(well, not so much)

Go Service

Load Balancer

Triton

Triton

Triton

...

Triton

Go Concurrency Primitives

“Go blue gopher as a primitive man”

WaitGroup

"Wait until these things are done"

WaitGroup

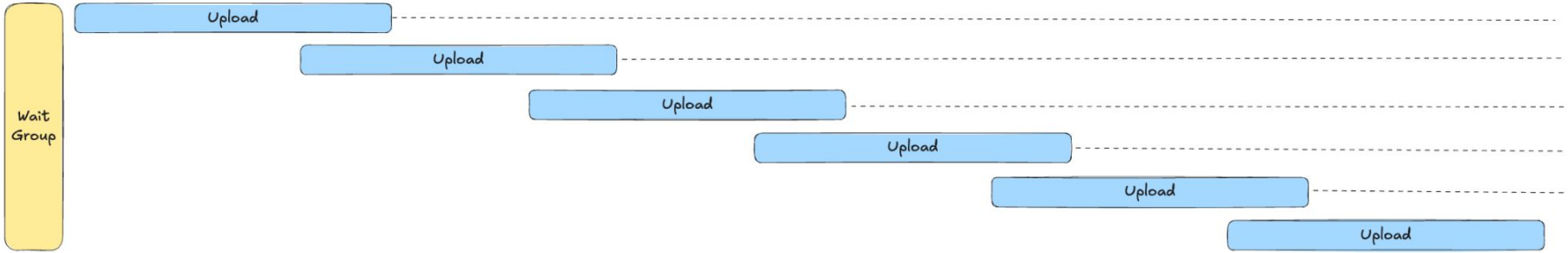
A **WaitGroup** waits for a collection of goroutines to finish.

- Maintains an internal counter
- **Add(delta int)** increments the counter
- **Done()** decrements the counter
- **Wait()** blocks until the counter is zero

WaitGroup

```go
func worker(id int, wg *sync.WaitGroup) {
  defer wg.Done() // Decrements counter when function returns

  fmt.Printf("Worker %d starting\n", id)
  time.Sleep(time.Second) // Simulate work
  fmt.Printf("Worker %d done\n", id)
}

func main() {
  var wg sync.WaitGroup

  // Launch 3 workers
  for i := 1; i <= 3; i++ {
    wg.Add(1) // Increment counter for each goroutine
    go worker(i, &wg)
  }

  // Wait for all goroutines to finish
  wg.Wait()
  fmt.Println("All workers completed")
}
```

WaitGroup



Semaphore

"Control how many things run at a time"

Semaphore

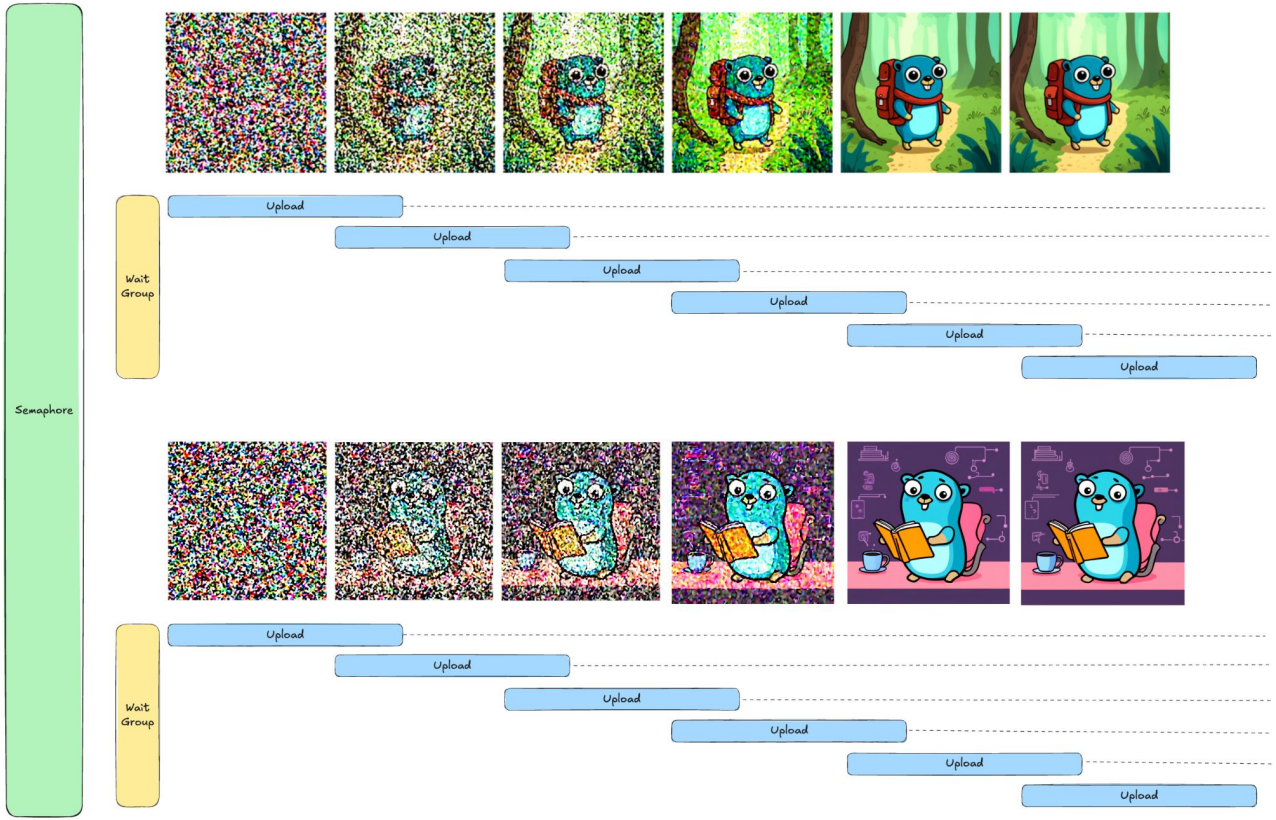
Semaphores are a concurrency primitive used to control access to shared resources.

- A semaphore maintains a count representing the number of available resources.
- **Acquire()** decrements the count, blocking if it's zero.
- **Release()** increments the count, potentially unblocking waiting goroutines.
- Semaphores can be used to limit the number of concurrent accesses to a resource.

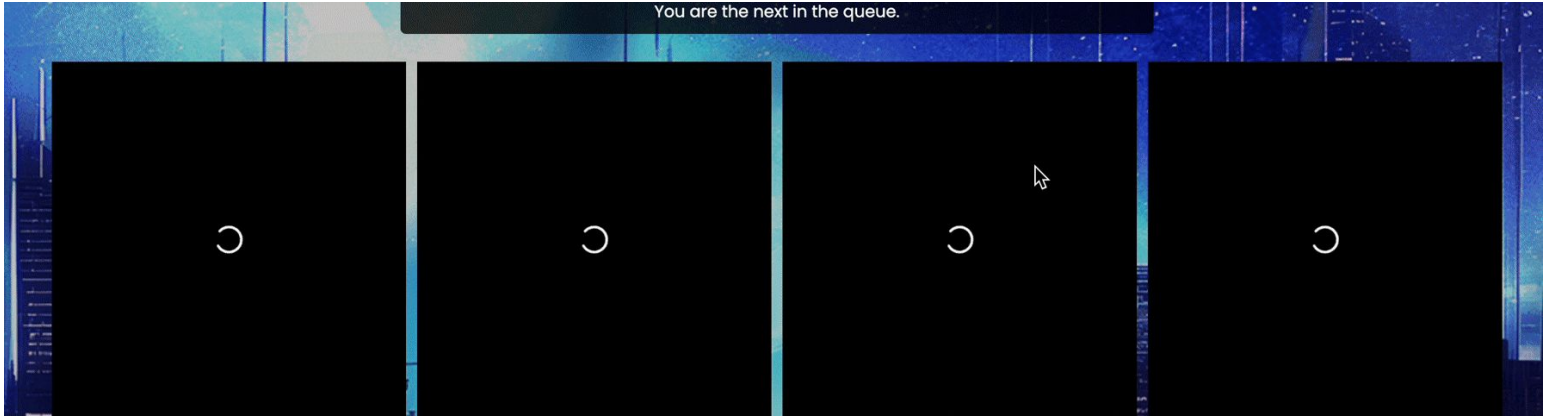
Semaphore

```go
func main() {
  // Create a semaphore with max weight of 3
  sem := semaphore.NewWeighted(3)
  ctx := context.Background()

  var wg sync.WaitGroup

  // Try to launch 5 workers, but only 3 can run concurrently
  for i := 1; i <= 5; i++ {
    wg.Add(1)
    go func(id int) {
      defer wg.Done()

      // Acquire semaphore
      if err := sem.Acquire(ctx, 1); err != nil {
        log.Printf("Failed to acquire semaphore: %v", err)
        return
      }
      defer sem.Release(1)

      fmt.Printf("Worker %d starting\n", id)
      time.Sleep(2 * time.Second) // Simulating work
      fmt.Printf("Worker %d done\n", id)
    }(i)
  }

  wg.Wait()
  fmt.Println("All workers completed")
}
```

Semaphore


Semaphore
Wait Group
Upload
Upload
Upload
Upload
Upload
Upload
Wait Group
Upload
Upload
Upload
Upload
Upload
Upload

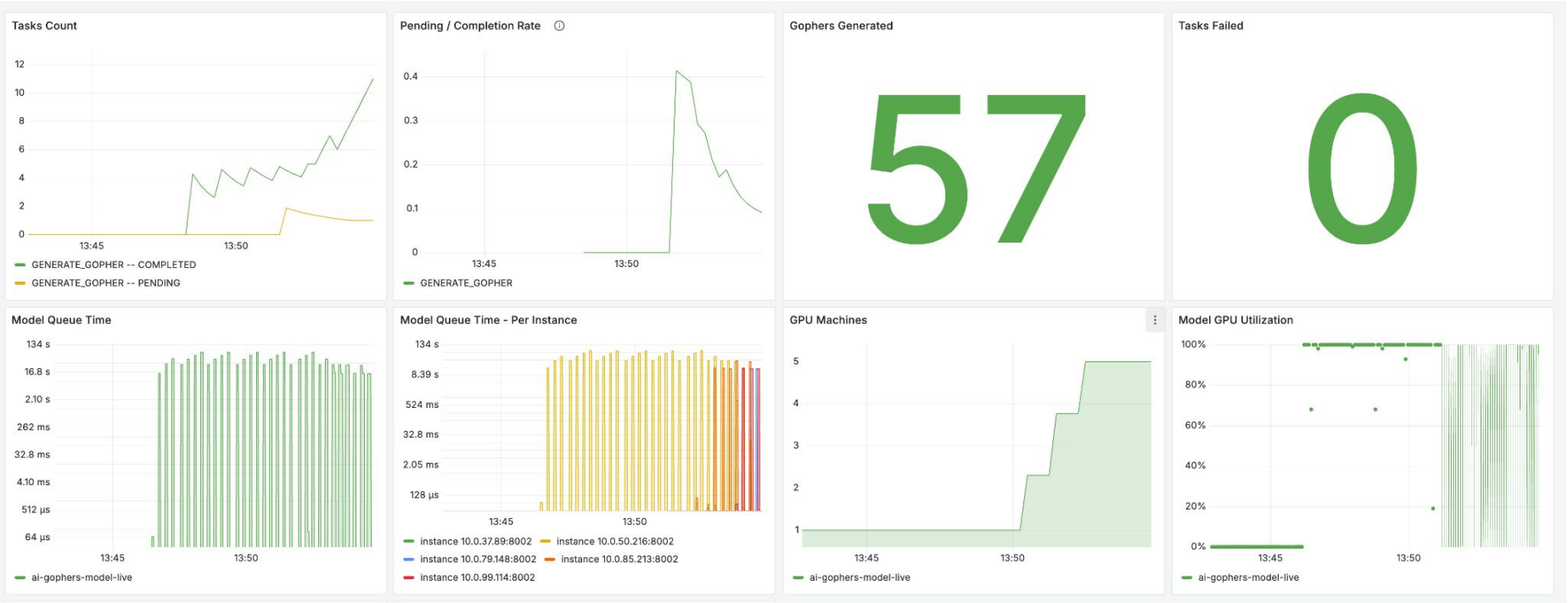
Final Result



A bit of observability

Observability

Observability



Last but not least

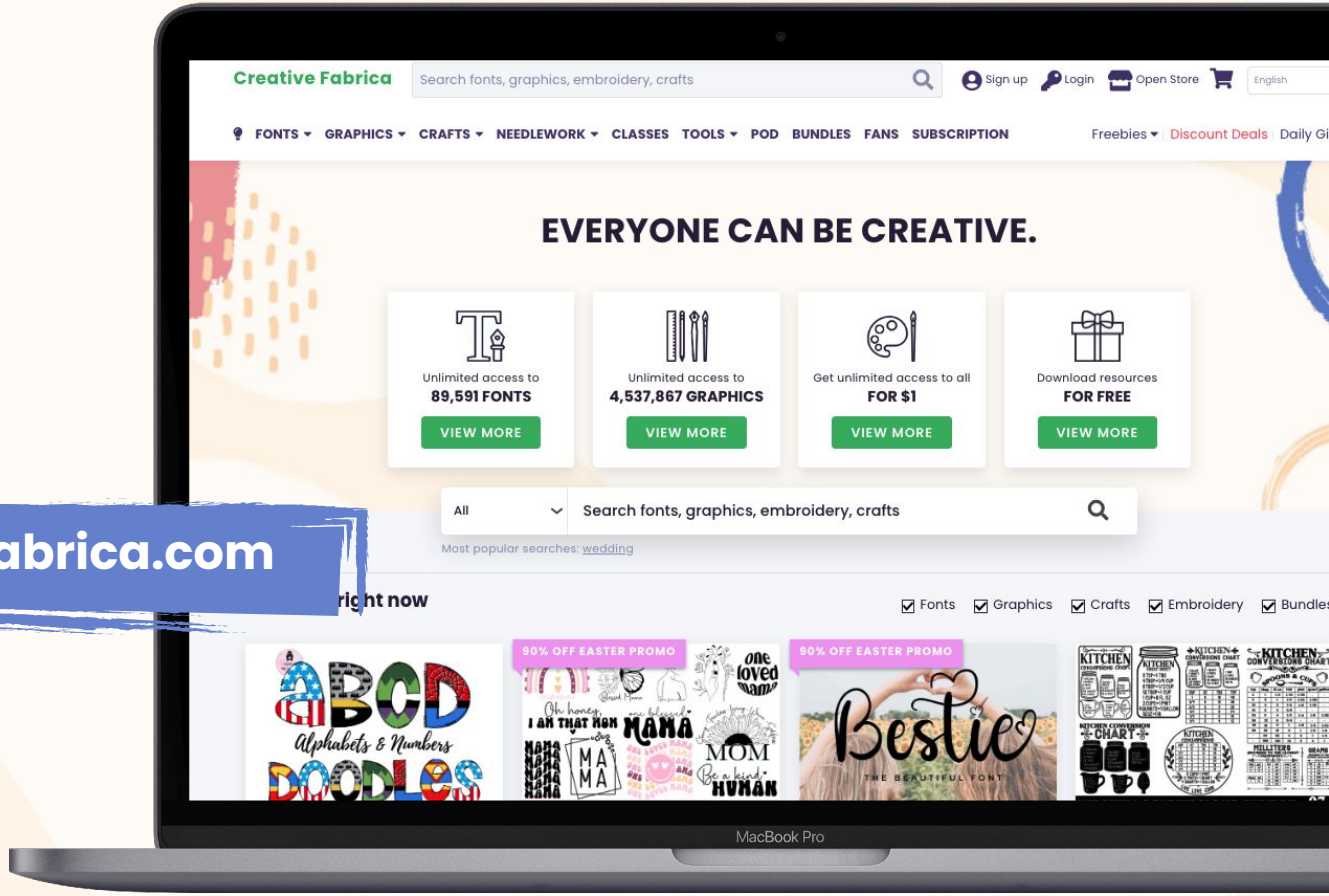
Unsung Heroes

Huge thanks to these guys! ❤️

Petr Byvshev
Data Scientist GenAI

Jonty De Klerk
Creative Lead

Ahmed Elsayed
Sr. Machine Learning Engineer

We're hiring!

careers.creativefabrica.com