LLM Advance Topic

Jiqiong QIU

Objectif du cours

- Le but de ce cours est l'introduction aux modèles LLM (large language model)
- Le cours se focalise essentiellement sur les aspects suivants :
  - *Evaluation LLM*
  - *Tuning LLM*
  - *Diffusion models*

Evaluation

How to craft the prompt template?

Is the model a good fit for the use case?

How to augment the model's knowledge?

Can tuning help?

Is the model performing over time?

Launch!

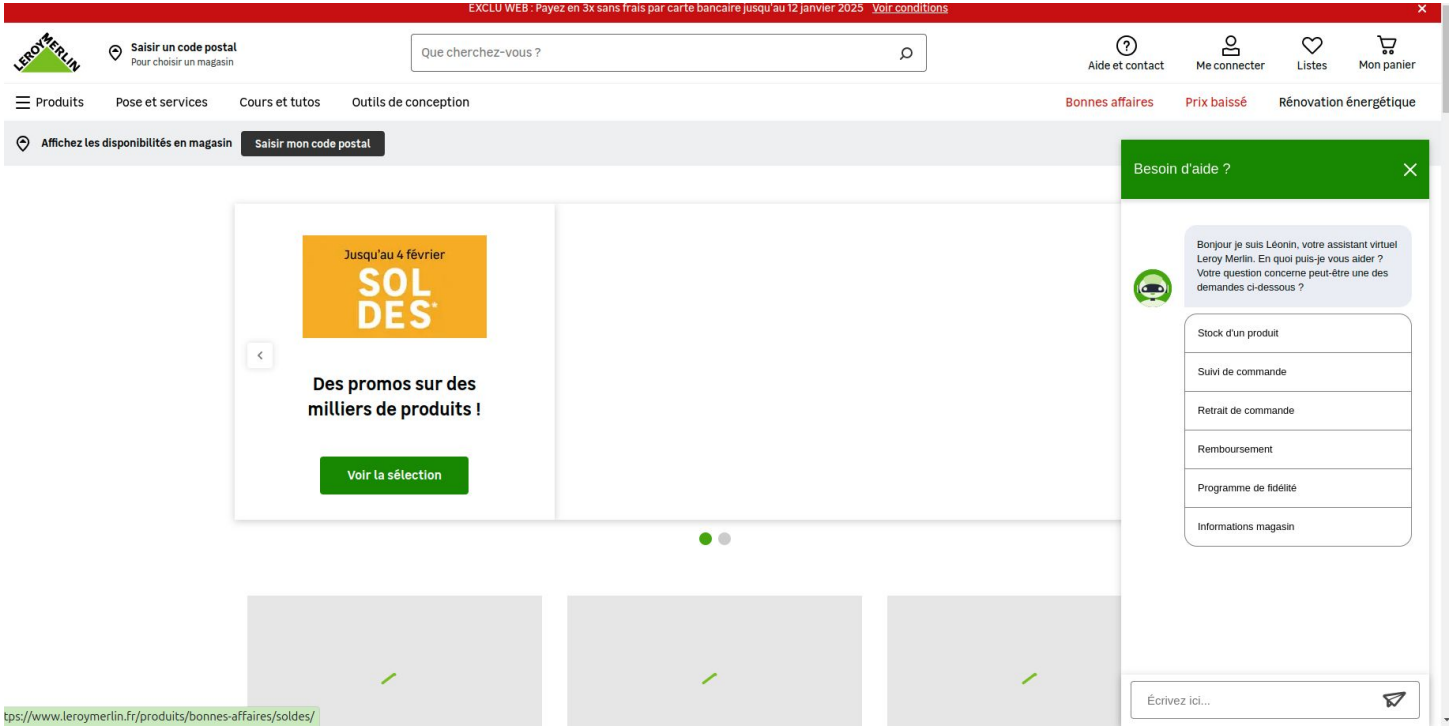
Pre-production

Production

Evaluation

GNOMON DIGITAL

Scenario: Leroy Merlin Chatbot (FAQ LLM)



Evaluation

GNOMON DIGITAL

1) Define evaluation goals
What specific task do you want the LLM to perform? Are you interested in overall fluency,coherence, factual accuracy and more?

2) Choose evaluation methods
Task-specific metrics, research benchmarks, LLM-based evaluations, human evaluations are available depending on the evaluation goals.

3) Select appropriate datasets
Define a “golden” dataset aligned with your evaluation goals and metrics. Consider benchmark datasets designed for LLMs evaluation.

4) Analyze and Interpret Results
Combine quantitative and qualitative results to derive evaluation insights. Take into account strengths and weaknesses evaluation methods and justify your conclusions.

Evaluation

GNOMON DIGITAL

1) Define evaluation goals

What specific task do you want the LLM to perform? Are you interested in overall fluency,coherence, factual accuracy and more?

2) Choose evaluation methods

Task-specific metrics, research benchmarks, LLM-based evaluations, human evaluations are available depending on the evaluation goals.

- Limited metrics
- sensitive to model
- Time and resource expensive

3) Select appropriate datasets

Define a “golden” dataset aligned with your evaluation goals and metrics. Consider benchmark datasets designed for LLMs evaluation.

- Lack of data
- Contaminated Data

4) Analyze and Interpret Results

Combine quantitative and qualitative results to derive evaluation insights. Take into account strengths and weaknesses evaluation methods and justify your conclusions.

- Lack of explainability
- Unsure what to do next

https://huggingface.co/docs/evaluate/choosing_a_metric

Evaluation : Computation Metrics

GNOMON®
DIGITAL

BLEU: https://huggingface.co/spaces/evaluate-metric/bleu

BLEU (Bilingual Evaluation Understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" – this is the central idea behind BLEU. BLEU was one of the first metrics to claim a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics.

Scores are calculated for individual translated segments—generally sentences—by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Neither intelligibility nor grammatical correctness are not taken into account.

ROUGE: https://huggingface.co/spaces/evaluate-metric/rouge

ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation.

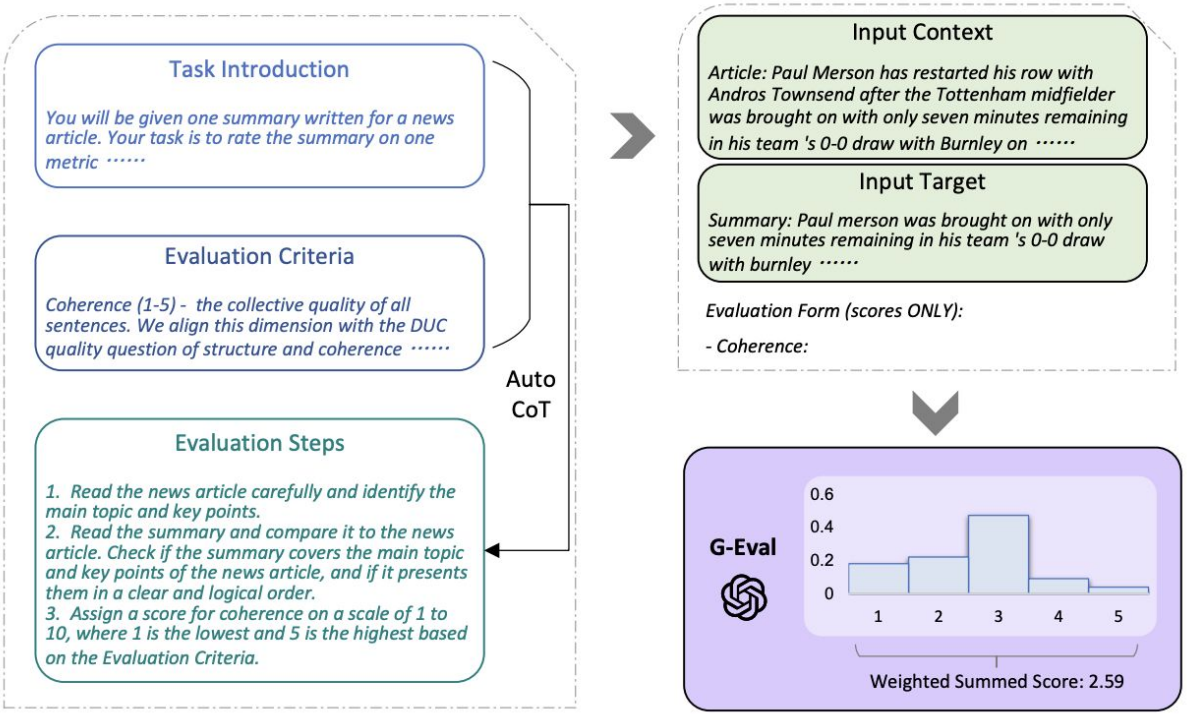
Note that ROUGE is case insensitive, meaning that upper case letters are treated the same way as lower case letters.

https://www.evidentlyai.com/llm-guide/llm-as-a-judge

Evaluation : Model-based Metrics

GNOMON DIGITAL

LLM as judge: Compare the performance of 2 models with an arbiter model



https://huggingface.co/docs/evaluate/choosing_a_metric

Evaluation

GNOMON DIGITAL

Computation-based metrics

- Use mathematical formulas to assess performance
- Comparison to ground truth is key
- Low cost and fast

Model-based metrics

- Use a judge model to assess performance based on descriptive evaluation criteria
- Ground truth is optional
- Slower and more expensive

Evaluation

Prepare evaluation dataset

```
instruction = "Summarize the following article"


context = [
    "To make a classic spaghetti carbonara, start by bringing a large pot of salted water to a boil. While the water is heating up, cook pancetta or guanciale in a skillet with olive oil over medium heat until it's crispy and golden brown. Once the pancetta is done, remove it from the skillet and set it aside. In the same skillet, whisk together eggs, grated Parmesan cheese, and black pepper to make the sauce. When the pasta is cooked al dente, drain it and immediately toss it in the skillet with the egg mixture, adding a splash of the pasta cooking water to create a creamy sauce.",
]
```

Evaluation

GNOMON DIGITAL

Prepare evaluation dataset

```
reference = [
    "The process of making spaghetti carbonara involves boiling pasta, crisping pancetta or guanciale, whisking together eggs and Parmesan cheese, and tossing everything together to create a creamy sauce.",
    "Preparing risotto entails sautéing onions and garlic, toasting Arborio rice, adding wine and broth gradually, and stirring until creamy and tender.",
]
#Create a dataframe with instruction, context and reference
eval_dataset = pd.DataFrame(
    {
        "context": context,
        "reference": reference,
        "instruction": [instruction] * len(context),
    }
)
```

Evaluation

GNOMON DIGITAL

Prepare evaluation dataset

```
reference = [
    "The process of making spaghetti carbonara involves boiling pasta, crisping pancetta or guanciale, whisking together eggs and Parmesan cheese, and tossing everything together to create a creamy sauce.",
    "Preparing risotto entails sautéing onions and garlic, toasting Arborio rice, adding wine and broth gradually, and stirring until creamy and tender.",
]
#Create a dataframe with instruction, context and reference
eval_dataset = pd.DataFrame(
    {
        "context": context,
        "reference": reference,
        "instruction": [instruction] * len(context),
    }
)
```

Evaluation

GNOMON DIGITAL

Perform evaluation

```
eval_task = EvalTask(
    dataset=eval_dataset,
    metrics=metrics,
    experiment=experiment_name,
)
```

https://github.com/GoogleCloudPlatform/asl-ml-immersion/blob/master/notebooks/vertex_genai/solutions/gen_ai_evaluation_service.ipynb

Evaluation

GNOMON DIGITAL

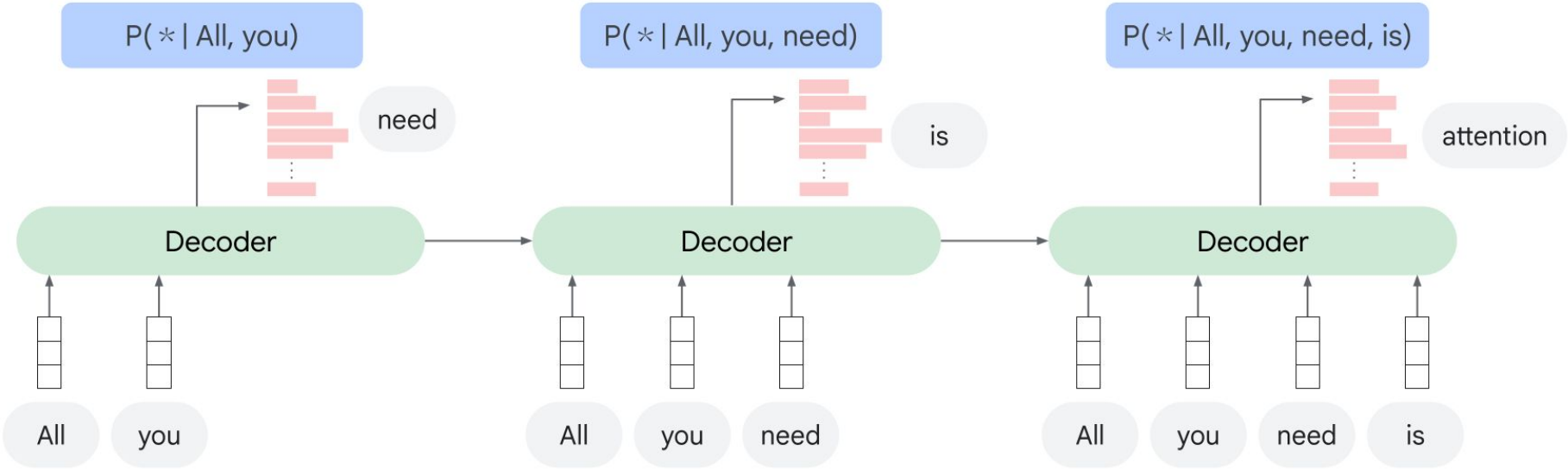
Perform evaluation

```
prompt_templates = [
    "Instruction: {instruction}. Article: {context}. Summary:",
    "Article: {context}. Complete this task: {instruction}, in one sentence. Summary:",
] #these are the prompt templates we want to evaluate

for i, prompt_template in enumerate(prompt_templates):
    experiment_run_name = f"eval-prompt-engineering-{run_id}-prompt-{i}"

    eval_result = eval_task.evaluate(
        prompt_template=prompt_template,
        experiment_run_name=experiment_run_name,
        model=gemini_model,
    )

    eval_results.append(
           (f"Prompt #{i}", eval_result.summary_metrics, eval_result.metrics_table)
    )
```

Fin-tuning LLM

GNOMON DIGITAL

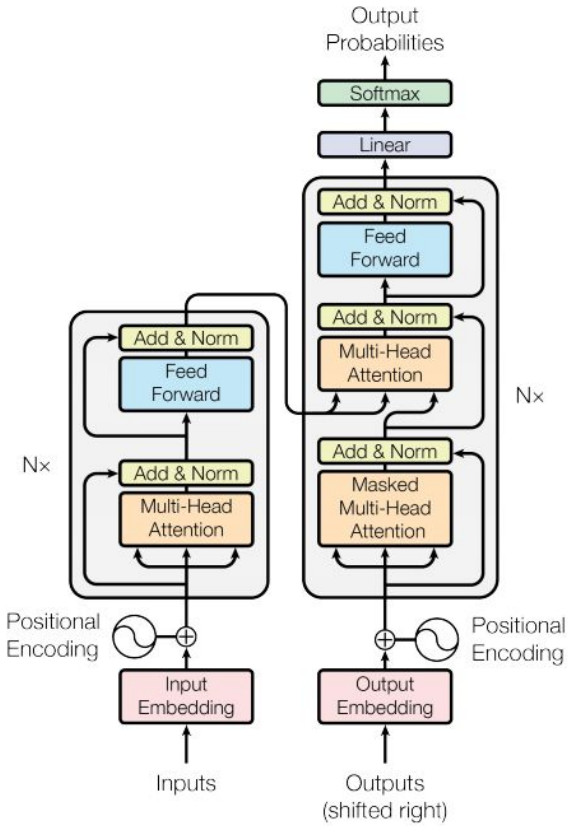
What is a language model?



Fin-tuning LLM

GNOMON DIGITAL

Keras NLP



Neural Networks: Zero to Hero

A course by Andrej Karpathy on building neural networks, from scratch, in code.
We start with the basics of backpropagation and build up to modern deep neural networks, like GPT. In my opinion language models are an excellent place to learn deep learning, even if your intention is to eventually go to other areas like computer vision because most of what you learn will be immediately transferable. This is why we dive into and focus on languade models.
Prerequisites: solid programming (Python), intro-level math (e.g. derivative, gaussian).

Learning is easier with others, come say hi in our Discord channel:

Syllabus

2h25m The spelled-out intro to neural networks and backpropagation: building micrograd
This is the most step-by-step spelled-out explanation of backpropagation and training of neural networks. It only assumes basic knowledge of Python and a vague recollection of calculus from high school.

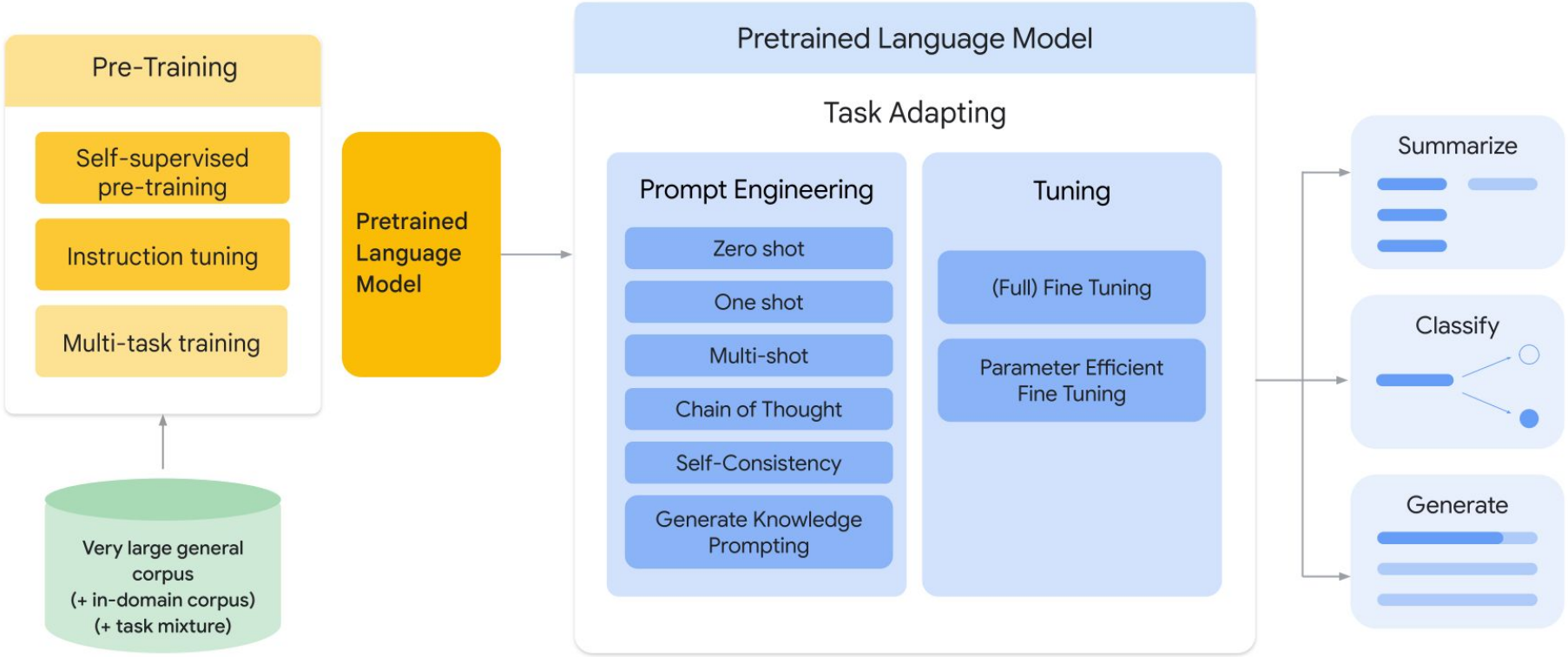
1h57m The spelled-out intro to language modeling: building makemore
We implement a bigram character-level language model, which we will further complexify in followup videos into a modern Transformer language model, like GPT. In this video, the focus is on (1) introducing torch.Tensor and its subtleties and use in efficiently evaluating neural networks and (2) the overall framework of language modeling that includes model training, sampling and the evaluation of a loss (e.g. the negative log likelihood for classification).

1h15m Building makemore Part 2: MLP
We implement a multilayer perceptron (MLP) character-level language model. In this video we also introduce many basics of machine learning (e.g. model training, learning rate tuning, hyperparameters, evaluation, train/dev/test splits, under/overfitting, etc.).

https://karpathy.ai/zero-to-hero.html

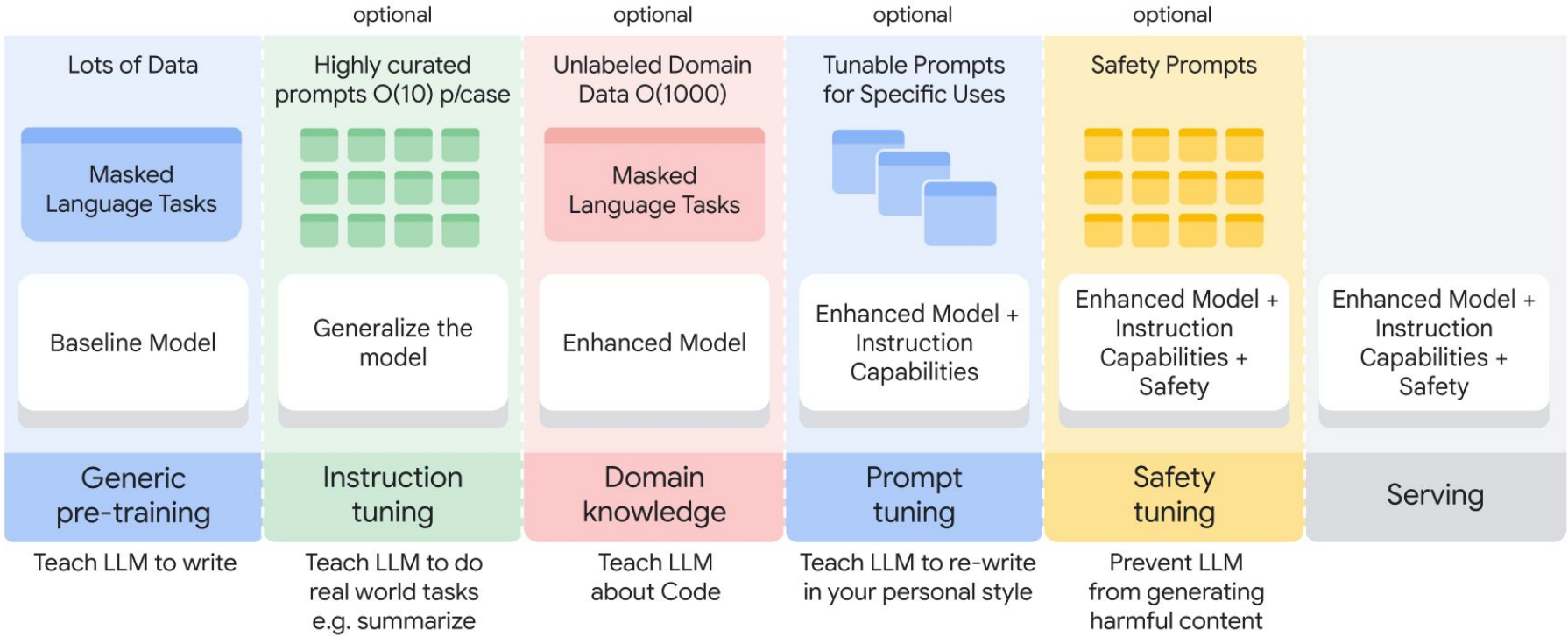
Fin-tuning LLM

GNOMON DIGITAL



Fin-tuning LLM

GNOMON DIGITAL



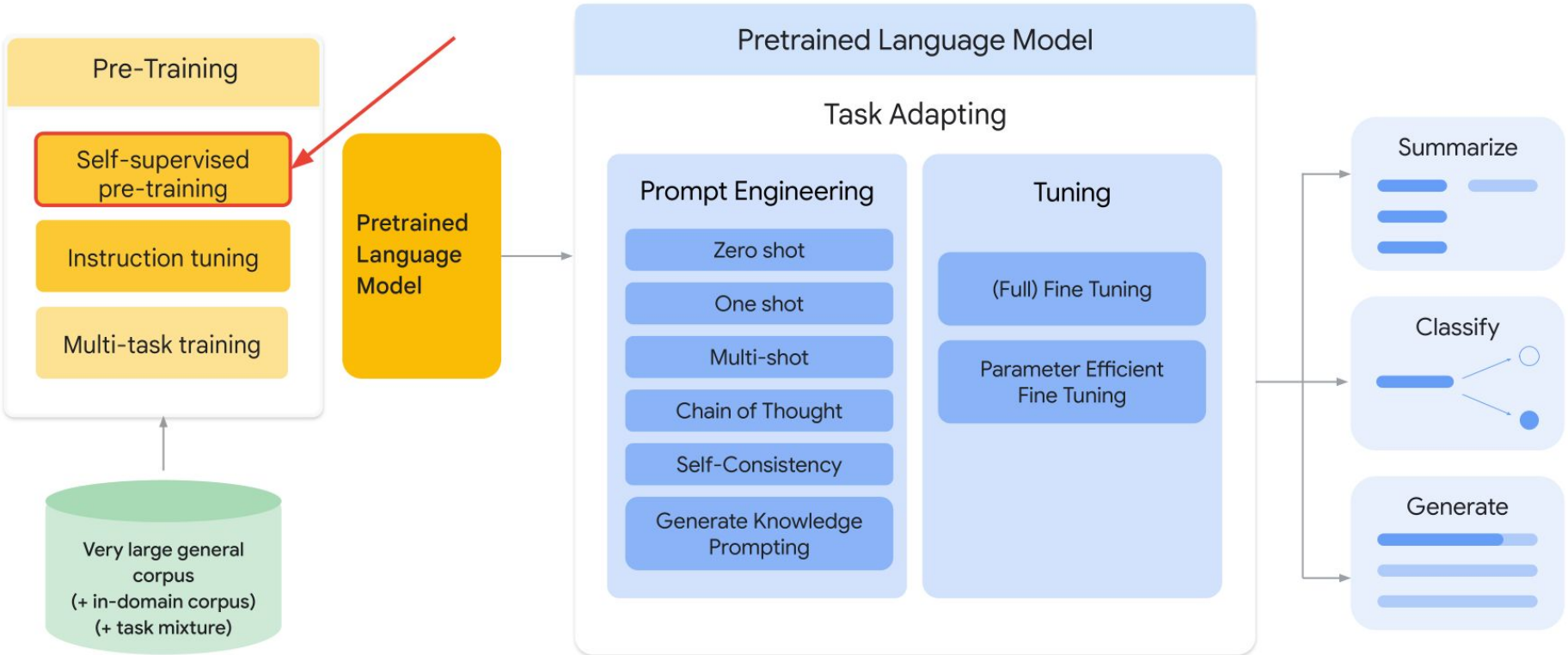
Fin-tuning LLM

GNOMON DIGITAL

Customization	Description	What gets updated?
Prompt Design	Create a prompt for a specific task	No updates to base model or weights
Prompt Tuning	Learn a prompt vector representation from data	Base model weights are frozen, but you learn a fixed size vector representation as an input for the task
Fine Tuning	Continued training on a specialized corpus	Update all of the base model weights

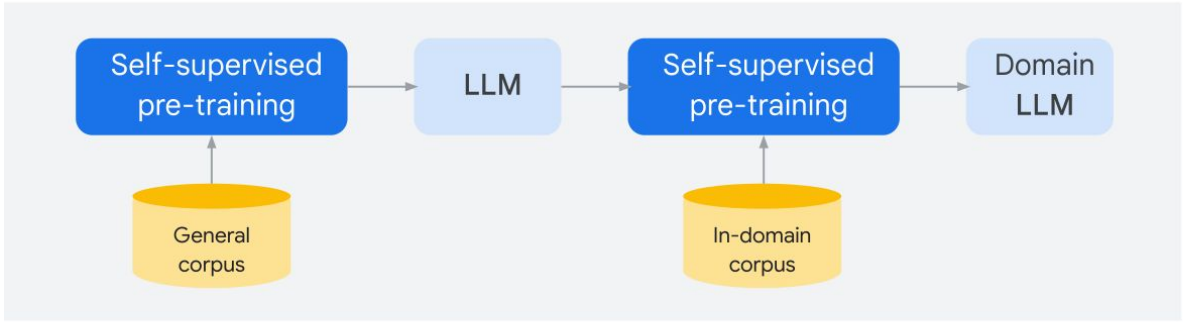
Fin-tuning LLM

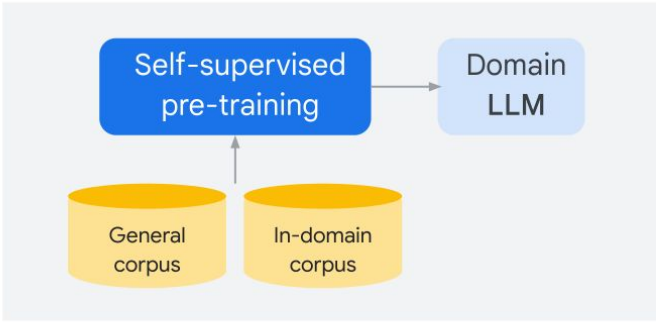
GNOMON DIGITAL

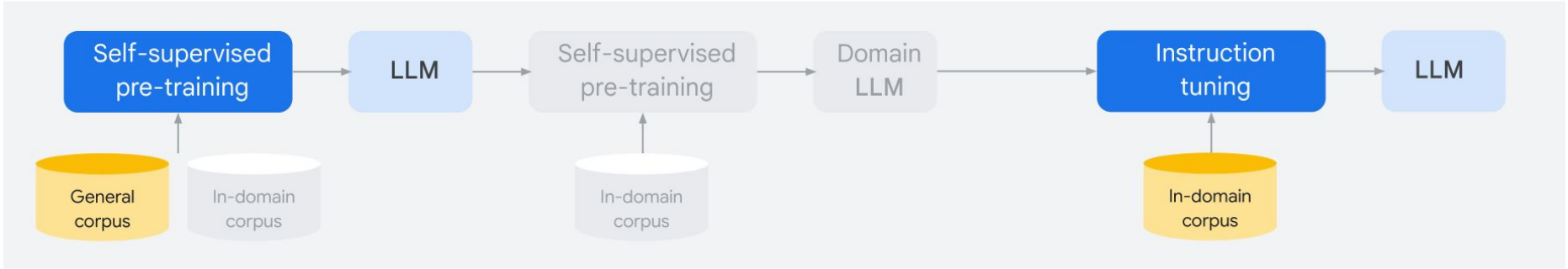


Fin-tuning LLM

GNOMON DIGITAL





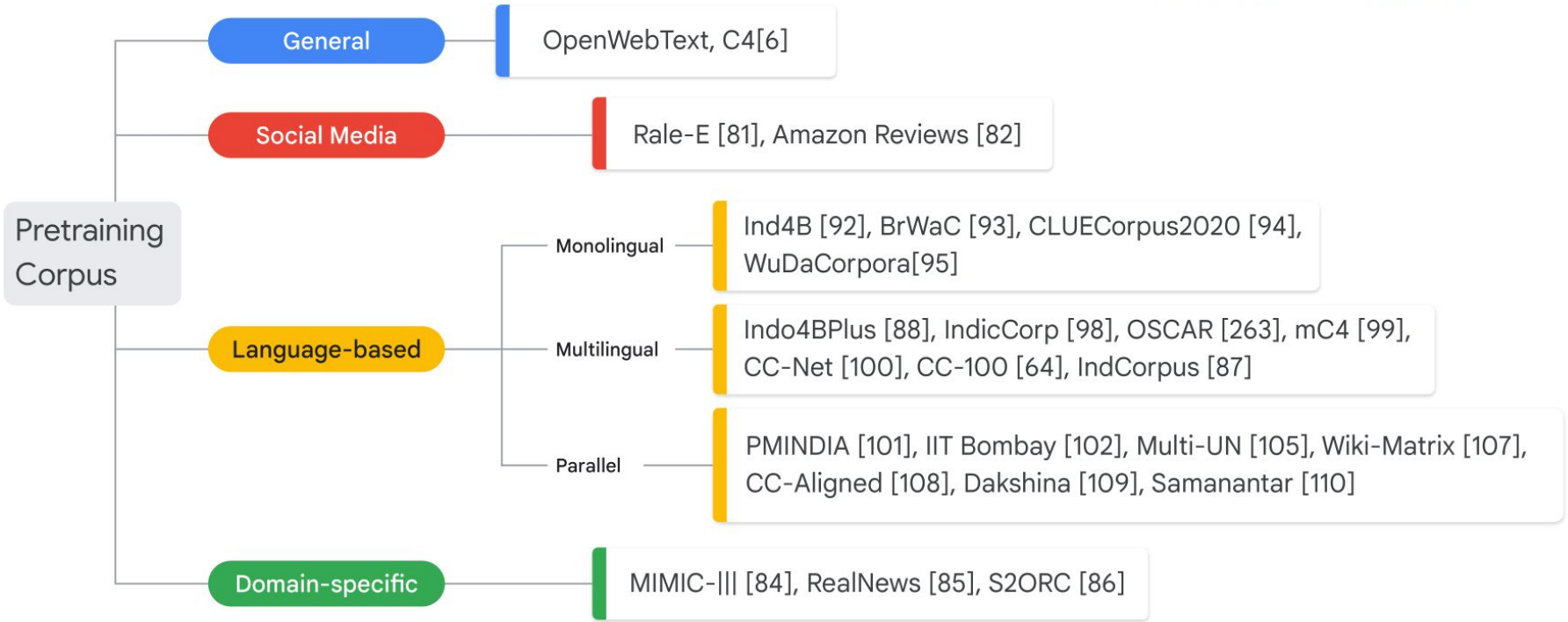


https://arxiv.org/pdf/2104.08691

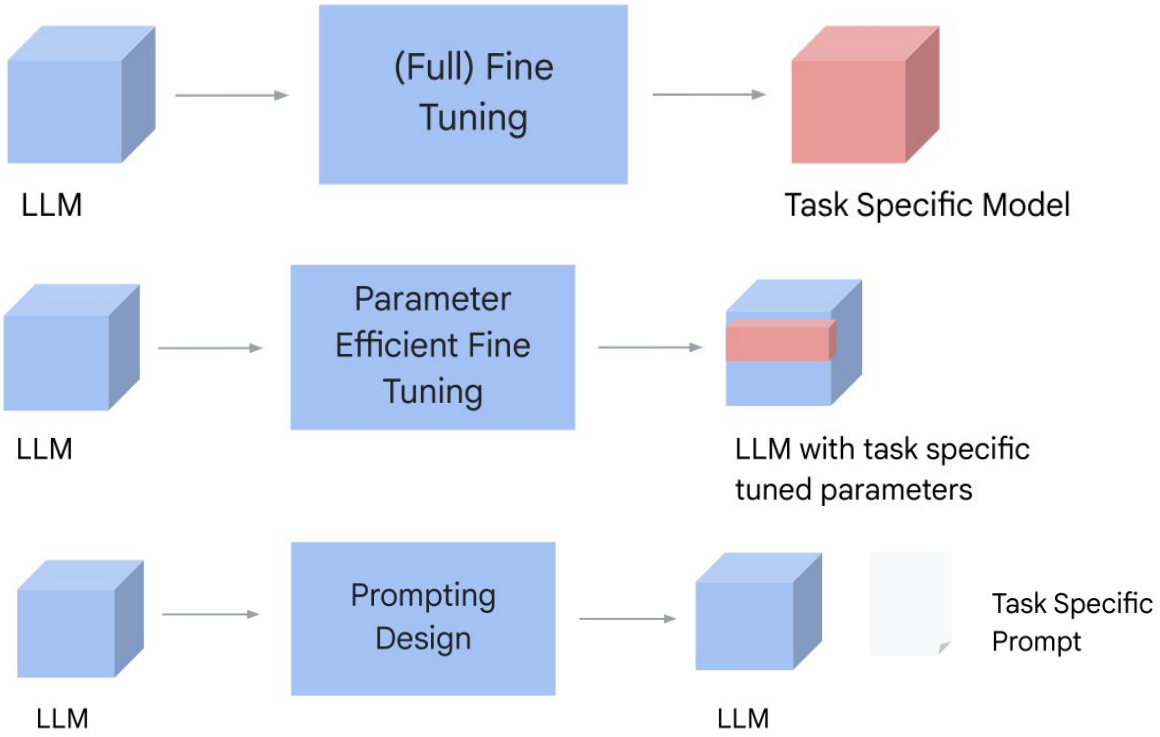
Fin-tuning LLM

The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* **Rami Al-Rfou** **Noah Constant**
Google Research
{brianlester,rmyeid,nconstant}@google.com



Fin-tuning LLM



https://arxiv.org/pdf/2204.05832

Fin-tuning LLM

GNOMON DIGITAL

What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization?

The BigScience Architecture & Scaling Group

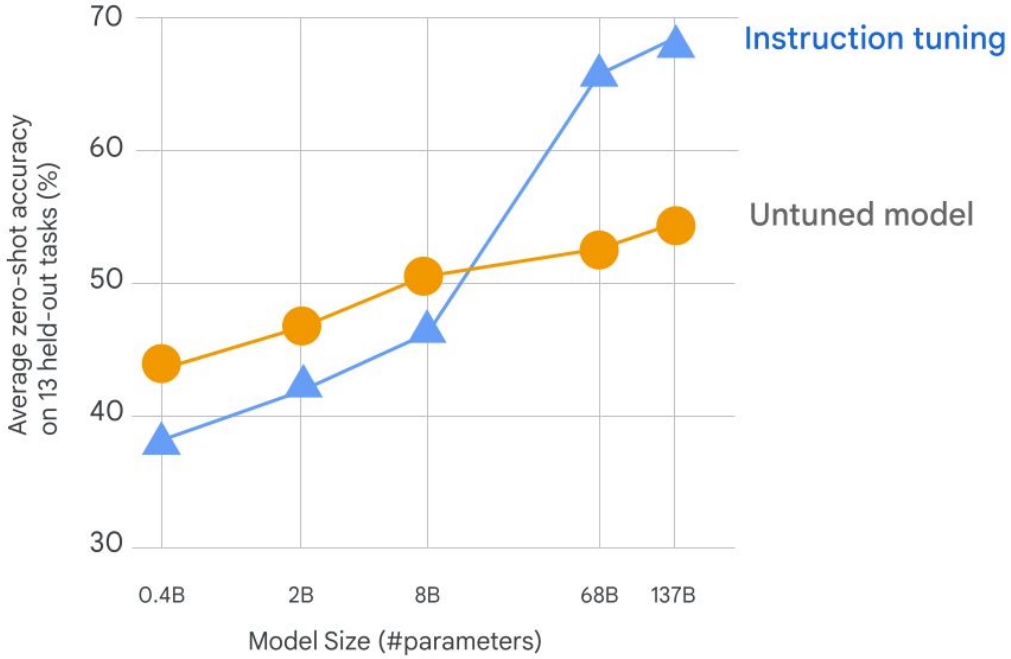
Thomas Wang[1*] Adam Roberts[2*]

Daniel Hesslow[3] Teven Le Scao[1] Hyung Won Chung[2]

Iz Beltagy[4] Julien Launay[3,5†] Colin Raffel[1†]

[1] Hugging Face [2]Google [3]LightOn

[4]Allen Institute for AI [5]LPENS, École Normale Supérieure



Fin-tuning LLM

GNOMON DIGITAL

Instruction Fine-Tuning

Premise
Russian cosmonaut Valery Polyakov set the record for the longest continuous amount of time spent in space, a staggering 438 days, between 1994 and 1995

Hypothesis
Russians hold the record for the longest stay in space

Target
Entailment
Not entailment

Options:
- Yes
- no

Template 1

<premise>
Based on the paragraph above, could we conclude that
<hypothesis>?
<options>

Template 2

<premise>
Can we infer the following?
<hypothesis>
<options>

Template 3

Read the following and determine if the hypothesis can be inferred from the premise:
Premise: <premise>
Hypothesis: <hypothesis>
<options>

Template 4

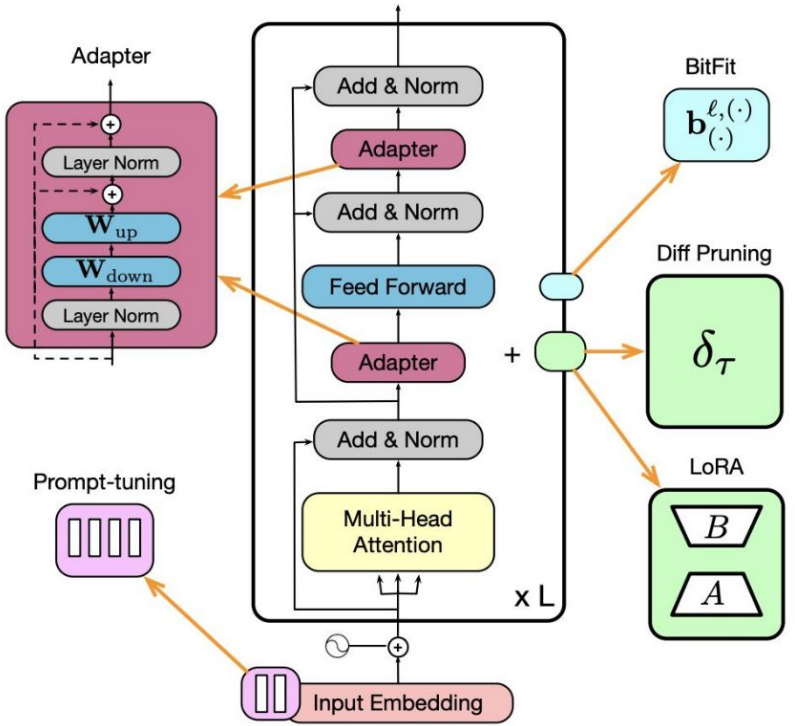
...

Fin-tuning LLM

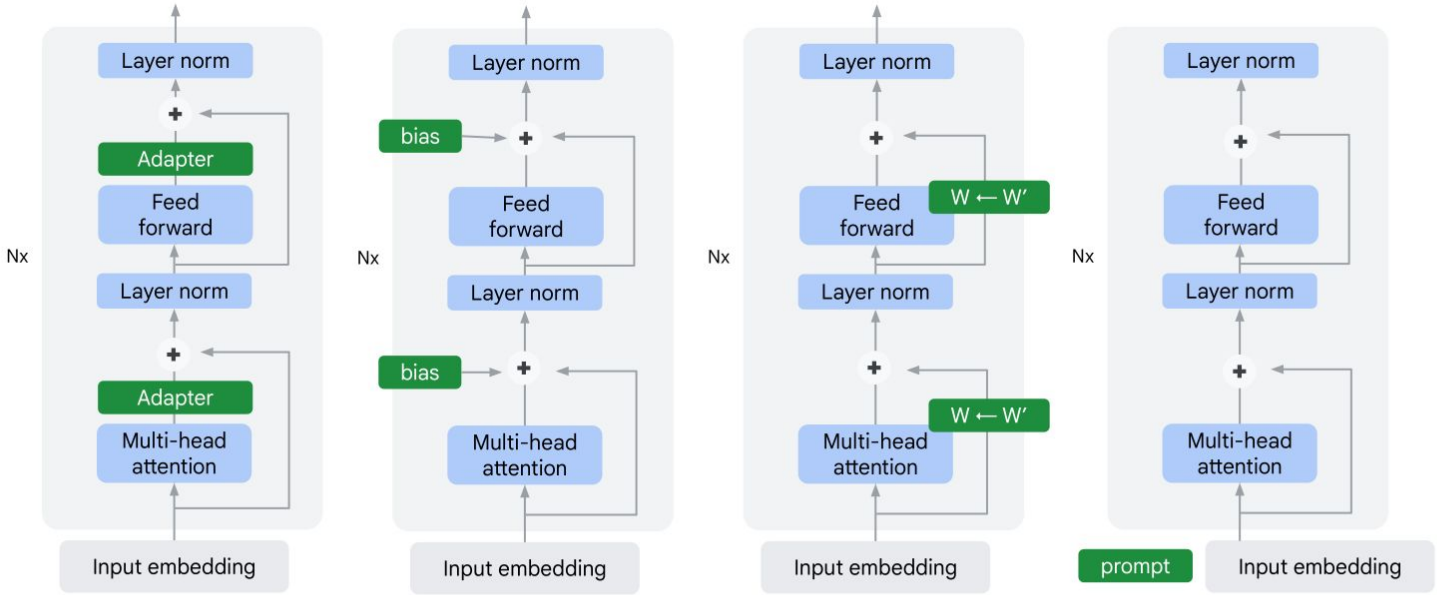
GNOMON DIGITAL

Parameter efficient tuning

- Model-Level
  - Adapter-Tuning
- Feature-Level
  - Prompt-Tuning
- Parameter-Level
  - LoRA
- Partial Fine-tuning
  - BitFit



Fin-tuning LLM



Adapters

BitFit

LoRA

$W' = W + AxB$

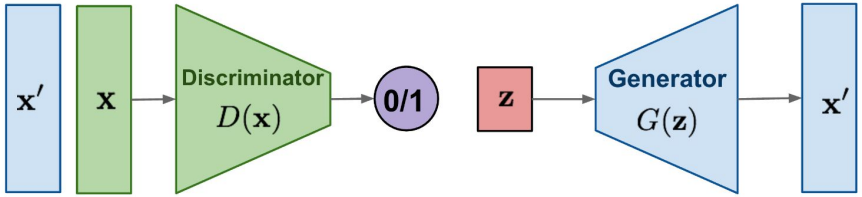
Prompt Tuning

Google Cloud

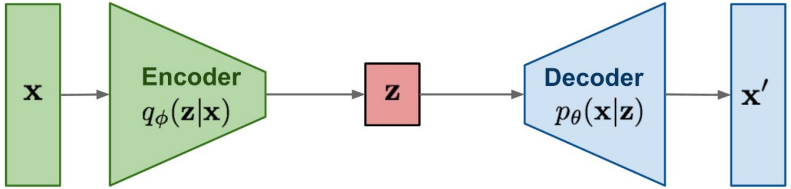
Diffusion Model

GNOMON DIGITAL

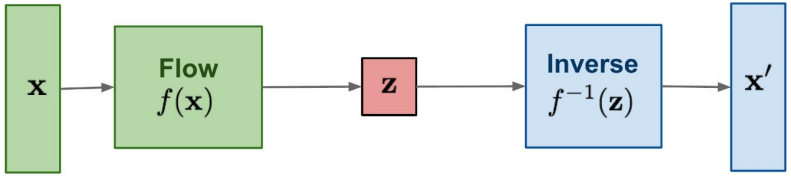
GAN: Adversarial training



VAE: maximize variational lower bound



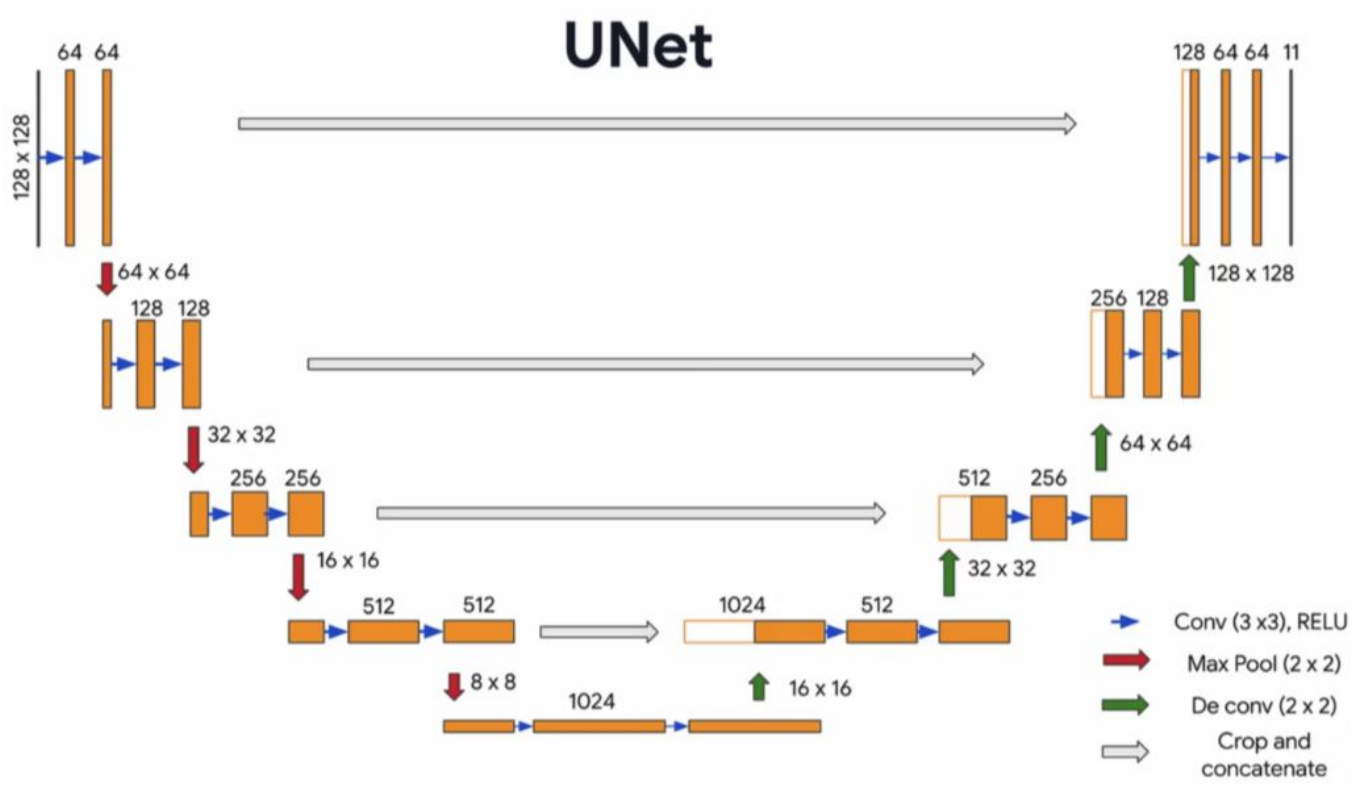
Flow-based models: Invertible transform of distributions



Diffusion models: Gradually add Gaussian noise and then reverse

Diffusion Model

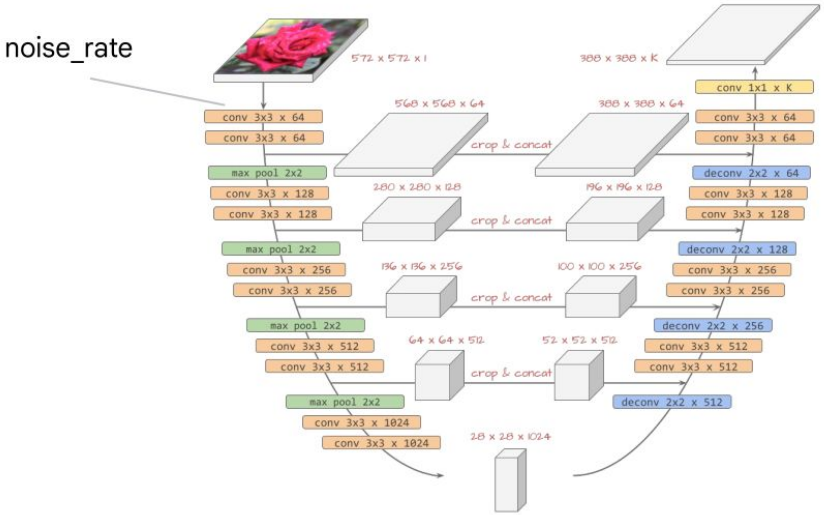
GNOMON DIGITAL

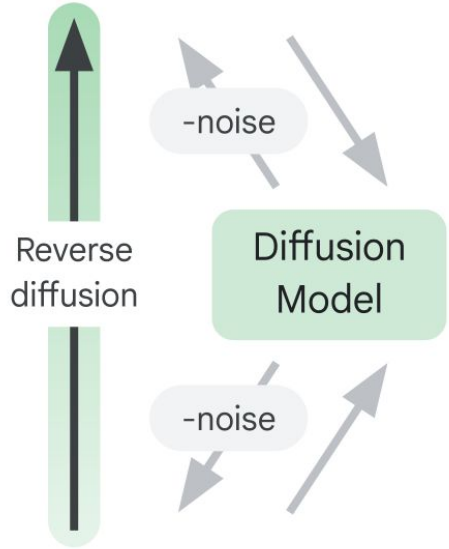


Diffusion Model

GNOMON DIGITAL

- U-Net architecture with residual connections
- Predicts noise in noisy image



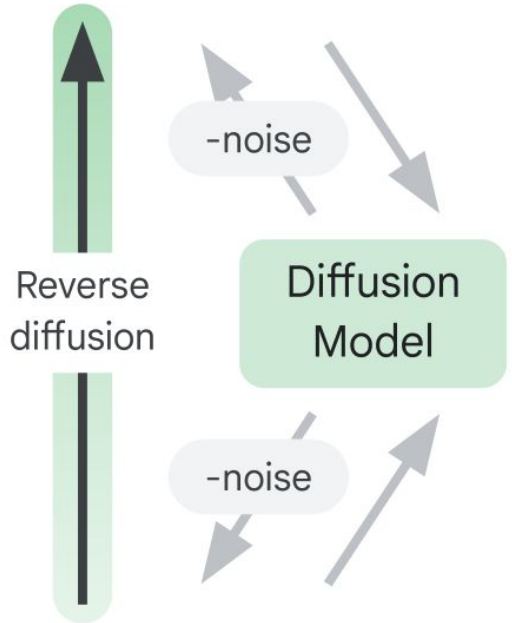


Diffusion Model

Our denoising model will be composed of

- 3 different layer blocks
  - Residual Block (residual connections)
  - Down Block (downsampling)
  - Up Block (upsampling)

- 1 custom Sin Embedding layer for embedding the noise_rate



Diffusion Model

GNOMON DIGITAL

Denoising Model - Noise Rate Embedding

```
class SinEmbedding(tf.keras.layers.Layer):
    def __init__(self, embedding_dim=32, min_freq=1.0, max_freq=1000.0, **kwargs):
        super().__init__(**kwargs)
        self.frequencies = tf.exp(
            tf.linspace(
                tf.math.log(min_freq),
                tf.math.log(max_freq),
                embedding_dim // 2
            )
        )
        self.angular_speeds = 2.0 * math.pi * self.frequencies

    def call(self, inputs):
        inputs = tf.cast(inputs, dtype=tf.float32)
        emb = tf.concat(
            [tf.sin(self.angular_speeds * inputs), tf.cos(self.angular_speeds * inputs)], axis=3
        )
        return emb
```

Allows model to learn how to denoise at different noise rates

Diffusion Model

GNOMON DIGITAL

Denoising Model - Residual Block

```
def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = tf.keras.layers.Conv2D(width, kernel_size=1)(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Conv2D(
            width, kernel_size=3, padding="same", activation=tf.keras.activations.swish
        )(x)
        x = tf.keras.layers.Conv2D(width, kernel_size=3, padding="same")(x)
        x = tf.keras.layers.Add()([x, residual])
        return x

    return apply
```

Channel dimensionality of input needs to be size of specified width

Diffusion Model

GNOMON DIGITAL

Denoising Model - Residual Block

```
def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = tf.keras.layers.Conv2D(width, kernel_size=1)(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Conv2D(
            width, kernel_size=3, padding="same", activation=tf.keras.activations.swish
        )(x)
        x = tf.keras.layers.Conv2D(width, kernel_size=3, padding="same")(x)
        x = tf.keras.layers.Add()([x, residual])
        return x

    return apply
```

Normalize and send through convolutional layers

Diffusion Model

GNOMON DIGITAL

Denoising Model - Residual Block

```
def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = tf.keras.layers.Conv2D(width, kernel_size=1)(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Conv2D(
            width, kernel_size=3, padding="same", activation=tf.keras.activations.swish
        )(x)
        x = tf.keras.layers.Conv2D(width, kernel_size=3, padding="same")(x)
        x = tf.keras.layers.Add()([x, residual])
        return x

    return apply
```

Add input to output of convolutional layers (residual connection)

Diffusion Model

GNOMON DIGITAL

Denoising Model - Down Block

```
def DownBlock(width, block_depth):
    def apply(x):
        x, skips = x
        for _ in range(block_depth):
            x = ResidualBlock(width)(x)
            skips.append(x)
        x = tf.keras.layers.AveragePooling2D(pool_size=2)(x)
        return x

    return apply
```

block_depth number of residual convolution blocks then average pooling to downsample

Diffusion Model

GNOMON DIGITAL

Denoising Model - Down Block

```
def DownBlock(width, block_depth):
    def apply(x):
        x, skips = x
        for _ in range(block_depth):
            x = ResidualBlock(width)(x)
            skips.append(x)
        x = tf.keras.layers.AveragePooling2D(pool_size=2)(x)
        return x

    return apply
```

Append output to list so we can add skips to upsampling

Diffusion Model

GNOMON DIGITAL

Denoising Model - Down Block

```
def DownBlock(width, block_depth):
    def apply(x):
        x, skips = x
        for _ in range(block_depth):
            x = ResidualBlock(width)(x)
            skips.append(x)
        x = tf.keras.layers.AveragePooling2D(pool_size=2)(x)
        return x

    return apply
```

Pooling to reduce dimensionality by factor of two (downsampling)

Diffusion Model

GNOMON DIGITAL

Denoising Model - Up Block

```
def UpBlock(width, block_depth):
    def apply(x):
        x, skips = x
        x = tf.keras.layers.UpSampling2D(size=2, interpolation="bilinear")(x)
        for _ in range(block_depth):
            x = tf.keras.layers.Concatenate()([x, skips.pop()])
            x = ResidualBlock(width)(x)
        return x

    return apply
```

Simple bilinear upsampling (could also be learned through transpose conv)

Diffusion Model

GNOMON DIGITAL

Denoising Model - Up Block

```
def UpBlock(width, block_depth):
    def apply(x):
        x, skips = x
        x = tf.keras.layers.UpSampling2D(size=2, interpolation="bilinear")(x)
        for _ in range(block_depth):
            x = tf.keras.layers.Concatenate()([x, skips.pop()])
            x = ResidualBlock(width)(x)
        return x

    return apply
```

Concat skip from downsampling layer of same dimensionality

Diffusion Model

GNOMON DIGITAL

Denoising Model - Up Block

```
def UpBlock(width, block_depth):
    def apply(x):
        x, skips = x
        x = tf.keras.layers.UpSampling2D(size=2, interpolation="bilinear")(x)
        for _ in range(block_depth):
            x = tf.keras.layers.Concatenate()([x, skips.pop()])
            x = ResidualBlock(width)(x)
        return x

    return apply
```

Residual convolutional block

Diffusion Model

GNOMON DIGITAL

Denoising Model

```
noisy_images = tf.keras.Input(shape=(image_size, image_size, image_channels))
noise_variances = tf.keras.Input(shape=(1, 1, 1))

e = SinEmbedding()(noise_variances)
e = tf.keras.layers.UpSampling2D(size=image_size, interpolation="nearest")(e)

x = tf.keras.layers.Conv2D(widths[0], kernel_size=1)(noisy_images)
x = tf.keras.layers.Concatenate()([x, e])

skips = []
for width in widths[:-1]:
    x = DownBlock(width, block_depth)([x, skips])

for _ in range(block_depth):
    x = ResidualBlock(widths[-1])(x)

for width in reversed(widths[:-1]):
    x = UpBlock(width, block_depth)([x, skips])

x = tf.keras.layers.Conv2D(image_channels, kernel_size=1, kernel_initializer="zeros")(x)
model = tf.keras.Model([noisy_images, noise_variances], x, name="residual_unet")
```

Diffusion Model

GNOMON DIGITAL

Putting it all together

Diffusion Model

```
class DiffusionModel(tf.keras.Model):
    def __init__(self, image_size, image_channels, widths, block_depth,
                 batch_size, min_signal_rate, max_signal_rate, ema, plot_diffusion_steps):
        super().__init__()
        self.image_size = image_size
        self.image_channels = image_channels
        self.batch_size = batch_size
        self.min_signal_rate = min_signal_rate
        self.max_signal_rate = max_signal_rate
        self.plot_diffusion_steps = plot_diffusion_steps
        self.ema = ema
        self.normalizer = tf.keras.layers.Normalization()
        self.network = get_network(image_size, image_channels, widths, block_depth)
        self.ema_network = tf.keras.models.clone_model(self.network)
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

```
class DiffusionModel(tf.keras.Model):
    def __init__(self, image_size, image_channels, widths, block_depth,
                 batch_size, min_signal_rate, max_signal_rate, ema, plot_diffusion_steps):
        super().__init__()
        self.image_size = image_size
        self.image_channels = image_channels
        self.batch_size = batch_size
        self.min_signal_rate = min_signal_rate
        self.max_signal_rate = max_signal_rate
        self.plot_diffusion_steps = plot_diffusion_steps
        self.ema = ema
        self.normalizer = tf.keras.layers.Normalization()
        self.network = get_network(image_size, image_channels, widths, block_depth)
        self.ema_network = tf.keras.models.clone_model(self.network)
```

De-noising model (U-Net)

Diffusion Model

GNOMON DIGITAL

Diffusion Model

```
class DiffusionModel(tf.keras.Model):

    def __init__(self, image_size, image_channels, widths, block_depth,
                 batch_size, min_signal_rate, max_signal_rate, ema, plot_diffusion_steps):
        super().__init__()
        self.image_size = image_size
        self.image_channels = image_channels
        self.batch_size = batch_size
        self.min_signal_rate = min_signal_rate
        self.max_signal_rate = max_signal_rate
        self.plot_diffusion_steps = plot_diffusion_steps
        self.ema = ema
        self.normalizer = tf.keras.layers.Normalization()
        self.network = get_network(image_size, image_channels, widths, block_depth)
        self.ema_network = tf.keras.models.clone_model(self.network)
```

For predictions only

GNOMON DIGITAL

Diffusion Model

```
class DiffusionModel(tf.keras.Model):
. . .

    def denoise(self, noisy_images, noise_rates, signal_rates, training):
        if training:
            network = self.network
        else:
            network = self.ema_network

        # predict noise component and calculate the image component using it
        pred_noises = network([noisy_images, noise_rates**2], training=training)
        pred_images = (noisy_images - noise_rates * pred_noises) / signal_rates

        return pred_noises, pred_images
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Random noise with the same shape as batch of images

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Sample uniform random diffusion times

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Create noisy images according to diffusion schedule

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Predict noise and image components of noisy image with denoising model (U-Net)

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

Diffusion Model

Compute loss between actual noise added to image and predicted noise

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Compute gradients of loss

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Use gradients to update weights of of denoising model (U-Net)

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL

Diffusion Model

Use gradients to update weights of of denoising model (U-Net)

```
class DiffusionModel(tf.keras.Model):
# . . .
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, self.image_size, self.image_size, self.image_channels))

        diffusion_times = tf.random.uniform(
            shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
        )
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:
            pred_noises, pred_images = self.denoise(
                noisy_images, noise_rates, signal_rates, training=True
            )

            noise_loss = self.loss(noises, pred_noises)  # used for training

        gradients = tape.gradient(noise_loss, self.network.trainable_weights)
        self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))
```

Diffusion Model

GNOMON DIGITAL