Alignment Engineering

Gopi Krishnan Rajbahadur

How to cite this session?

```
@misc{Rajbahadur2024AIwareTutorial,
author = {Gopi Krishnan Rajbahadur and Ahmed E. Hassan},
title = {Alignment Engineering},
howpublished = {Tutorial presented at the AIware Leadership Bootcamp 2024},
month = {November},
year = {2024},
address = {Toronto, Canada},
note = {Part of the AIware Leadership Bootcamp series.},
url = {https://aiwarebootcamp.io/slides/2024_aiwarebootcamp_rajbahadur_alignmentengineering.pdf }}
```

Overview of the session

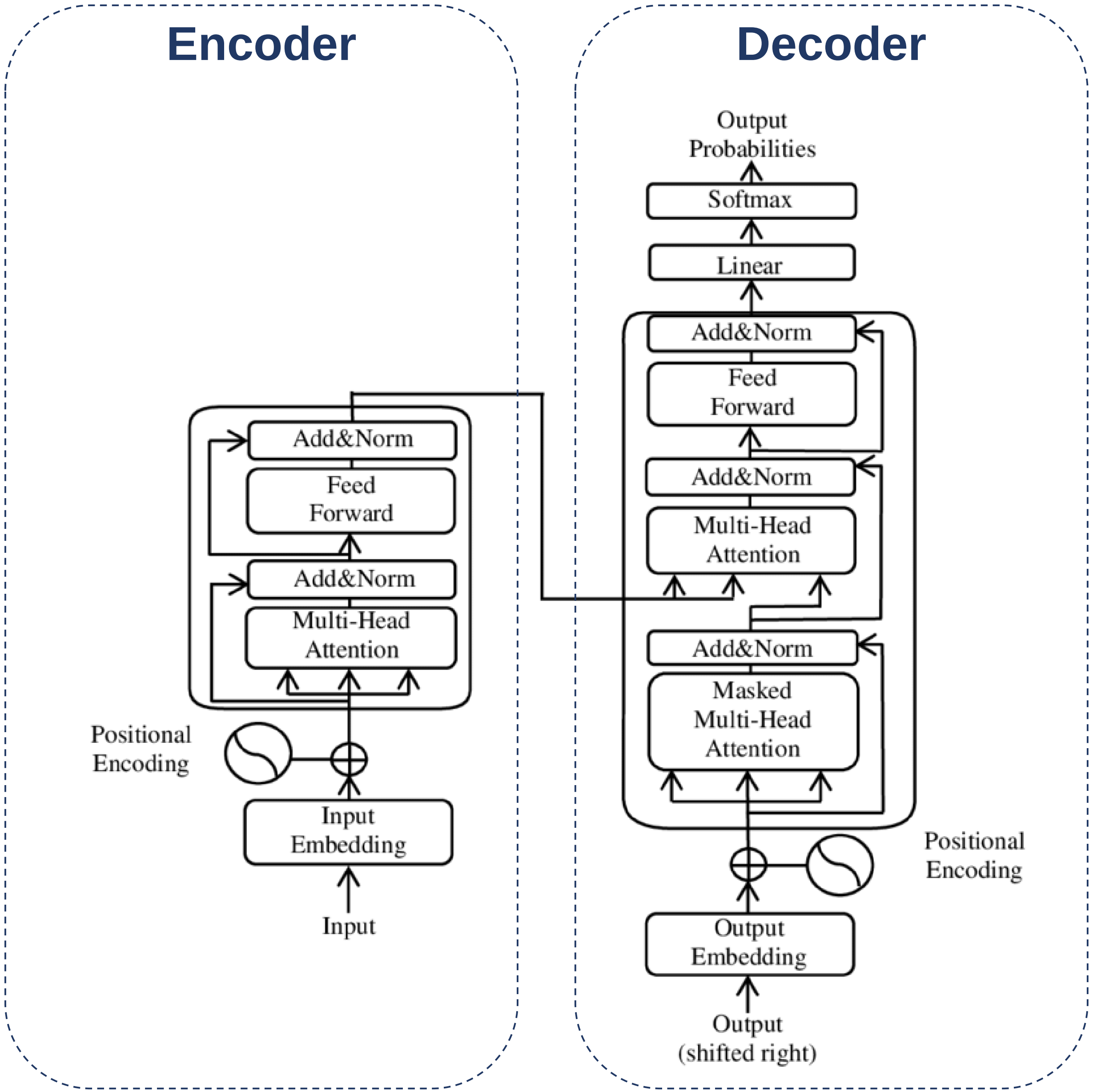
- ❑ **A brief intro to pre-training Foundation Models (FM):** An introduction to how a FM is pre-trained
- ❑ **Why do we have to align FMs:** Motivating the need for Alignment
- ❑ **Taxonomy of Alignment Engineering**
  - ❑ Data Alignment
  - ❑ Model Alignment
    - ❑ Finetuning
    - ❑ Online alignment
    - ❑ Offline alignment
  - ❑ Alignment Evaluation
- ❑ **Curriculum Learning**

Overview of the session

- ❑ **A brief intro to pre-training Foundation Models (FM):** An introduction to how a FM is pre-trained
- ❑ **Why do we have to align FMs:** Motivating the need for Alignment
- ❑ **Taxonomy of Alignment Engineering**
  - ❑ Data Alignment
  - ❑ Model Alignment
    - ❑ Finetuning
    - ❑ Online alignment
    - ❑ Offline alignment
  - ❑ Alignment Evaluation
- ❑ **Curriculum Learning**

It all started with Transformers

It all started with Transformers



Transformers: The Pivotal Shift Leading to Modern Foundation Models

Parallel Processing Advantage: Unlike RNNs, Transformers process sequences in parallel, significantly boosting training speed and efficiency.

Effective Long-Range Dependencies: Self-attention captures relationships across distant words better than RNNs, which struggle with long sequences.

Higher Scalability: Transformers scale effectively with more layers and parameters, enabling better performance on complex tasks.

Versatile Fine-Tuning: Pre-trained Transformers adapt easily to new tasks with minimal data, making them more flexible than RNNs across applications.

Types of transformers

ChatGPT

ENCODER ONLY

auto-encoding models

TASKS

- Sentence classification
- Named entity recognition
- Extractive question-answering
- Masked language modeling

EXAMPLES

BERT, RoBERTa, distilBERt

DECODER ONLY

auto-regressive models

TASKS

- Text generation
- Causal language modeling

EXAMPLES

GPT-2, GPT Neo, GPT-3

ENCODER DECODER

sequence-to-sequence models

TASKS

- Translation
- Summarization
- Generative question answering

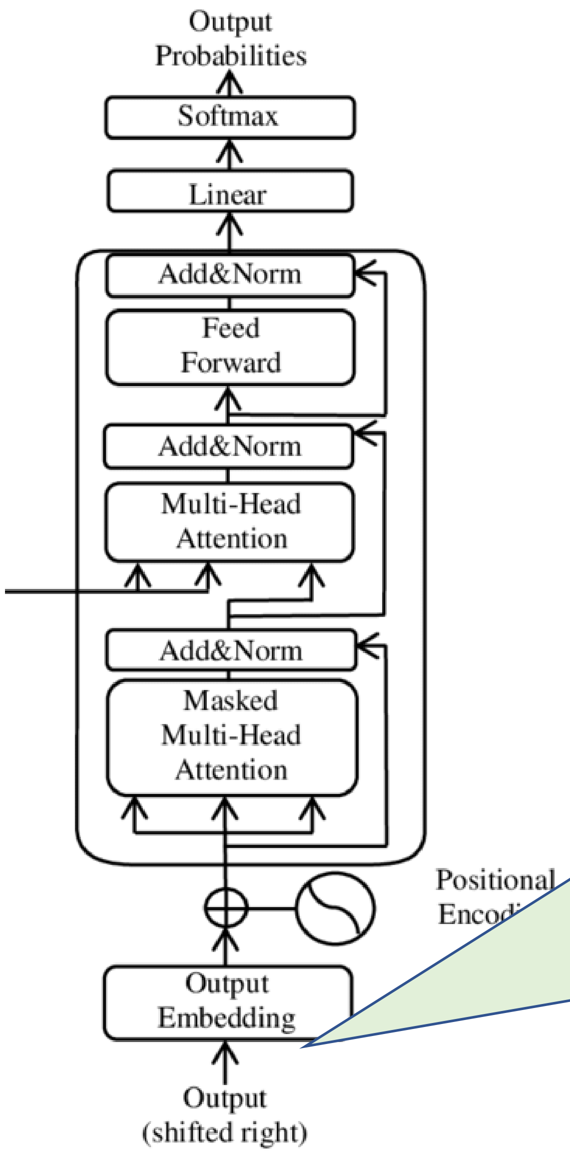
EXAMPLES

BART, T5, Marian

https://www.linkedin.com/pulse/introduction-transformers-arimac-ai-9r2rc/

Training a transformer - How do Decoder-only Transformers work?

First, the inputs are tokenized



Training paradigm: Self-supervision

Task: Next token prediction

Target Sequence: AIware Leadership Bootcamp is Awesome (many sentences like this)

Input Sequence: [START] AIware Leadership Bootcamp is [MASK]

Tokenization: Byte Pair Encoding (BPE)

1.Start with Characters:

- "AIware" → [A, I, w, a, r, e]
- "Leadership" → [L, e, a, d, e, r, s, h, i, p]
- "awesome" → [a, w, e, s, o, m, e]

2.Merge Frequent Pairs:

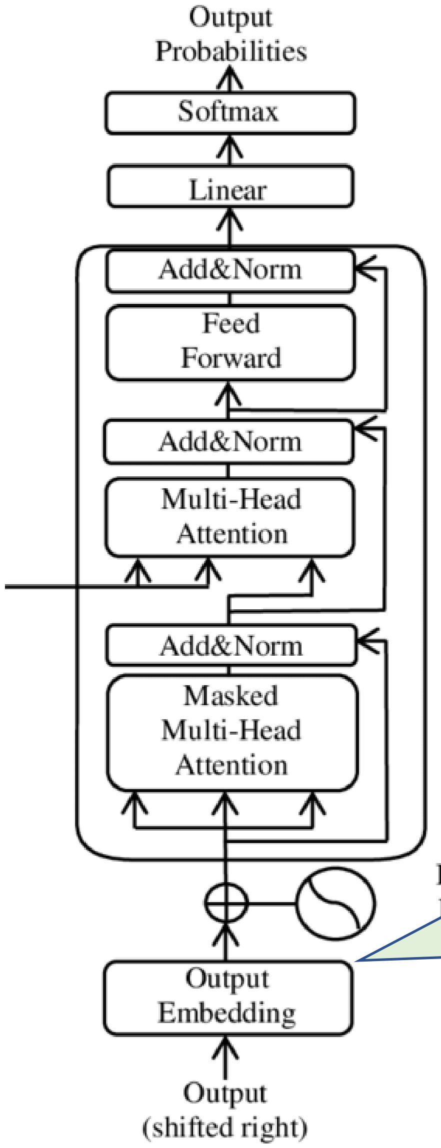
- Iteration 1: Merge "AI," "Le," "oo," "aw"
- Iteration 2: Merge "AIware," "Lea," "som"

3.Final Tokens:

- [AIware], [Leadership], [Bootcamp], [is], [awesome]
- [AIware], [Leadership], [Bootcamp], [is],[MASK]

Training a transformer - How do Decoder-only Transformers work?

Second, we embed the inputs



Tokenization & Indexing: Each word in "AIware Leadership Bootcamp is awesome" is tokenized and assigned an index:

> "AIware" → index 1243, "Leadership" → index 753, etc.

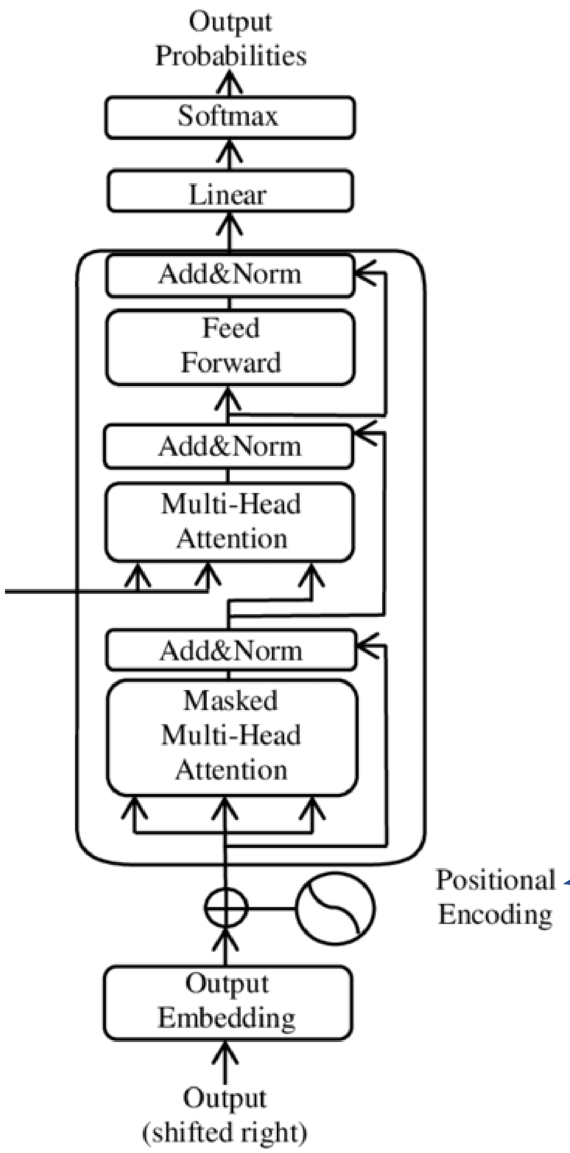
Embedding Matrix Lookup: An embedding matrix converts each token index into a **768-dimensional vector (common in models like GPT)**:

> "AIware" → [0.25, -0.13, 0.77, ..., 0.02]
> "Leadership" → [0.15, 0.56, -0.34, ..., 0.81]

Purpose of Output Embedding: These vectors provide initial representations of each token, containing semantic information without position or context.

Training a transformer - How do Decoder-only Transformers work?

Third, feed the positional information



Need for Positional Encoding: Transformers process all tokens in parallel, so they need **positional information** to understand the sequence order.

Adding Position Vectors: Each position (1, 2, 3, …) has a unique **positional encoding vector** (768 dimensions).

For example:

Position 1 (AIware) → [0.1, 0.2, 0.3, ..., 0.9]

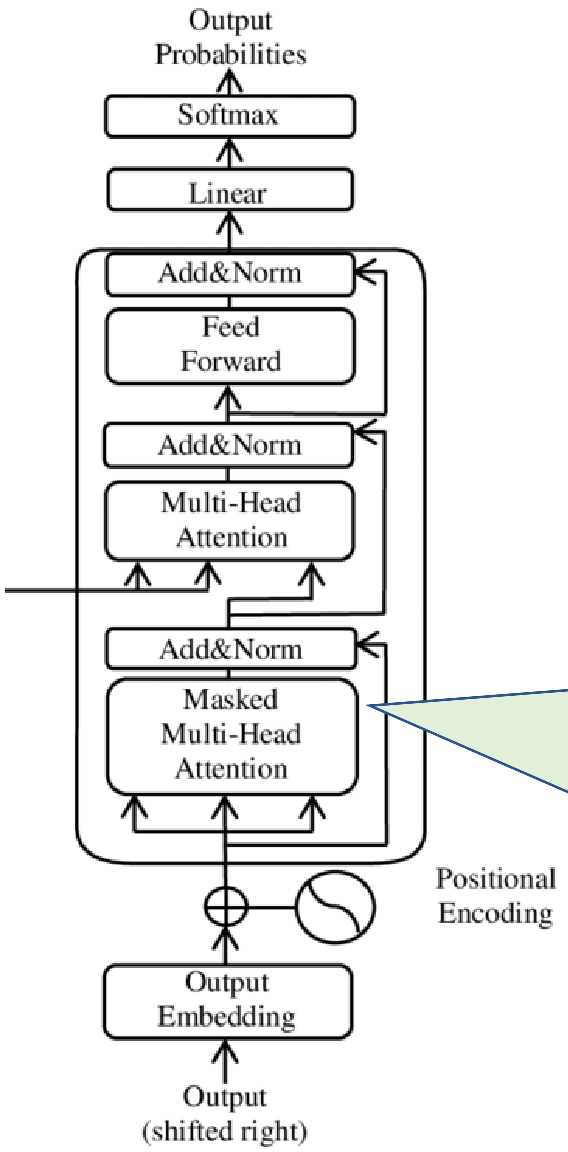
Position 2 (Leadership) → [0.15, 0.25, 0.35, ..., 0.95]

Combining Embedding + Position: The **positional vector** is added to each token's **output embedding**:

"AIware" final vector = embedding + position vector.

Training a transformer - How do Decoder-only Transformers work?

Fourth, Attention is all you need!



Purpose: Self-attention helps each token focus on re words in the sentence, enhancing context and meaning.

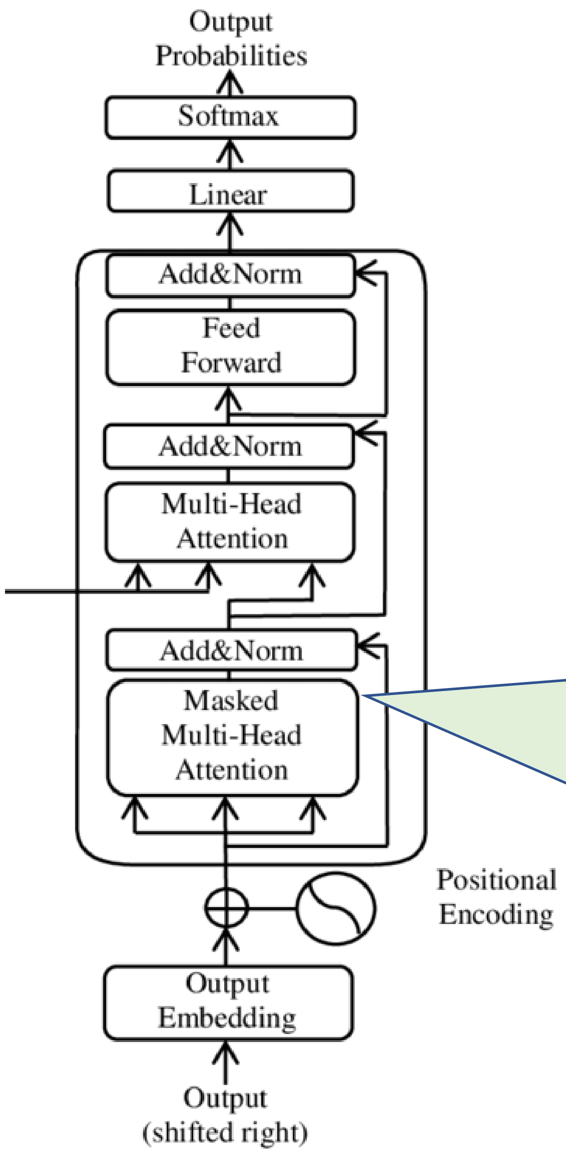
Query, Key, Value Vectors: Each token in "AIware Leadership Bootcamp is awesome" is transformed into Query (Q), Key (K), and Value (V) vectors using learned weight matrices ($\mathbf{W^Q}$, $\mathbf{W^K}$, $\mathbf{W^V}$)
Example: "AIware" → **Q, K, V** vectors via $\mathbf{W^Q}$, $\mathbf{W^K}$, $\mathbf{W^V}$

Calculating Attention Scores: Each word's Q vector (e.g., "Bootcamp") interacts with every other word's K vector to compute attention scores. Scores are softmaxed to produce attention weights, determining the relevance of each word pair.

Weighted Sum of Values: Each token's final representation is a weighted sum of Value vectors, where attention weights are applied to V vectors. For "awesome," this incorporates context from "AIware Leadership Bootcamp is."

Training a transformer - How do Decoder-only Transformers work?

Fourth, Lots of attention is the secret sauce



Purpose: Multi-head attention captures multiple relationships per word (e.g., Syntax and Semantics). Masking enforces a left-to-right sequence by hiding future words.

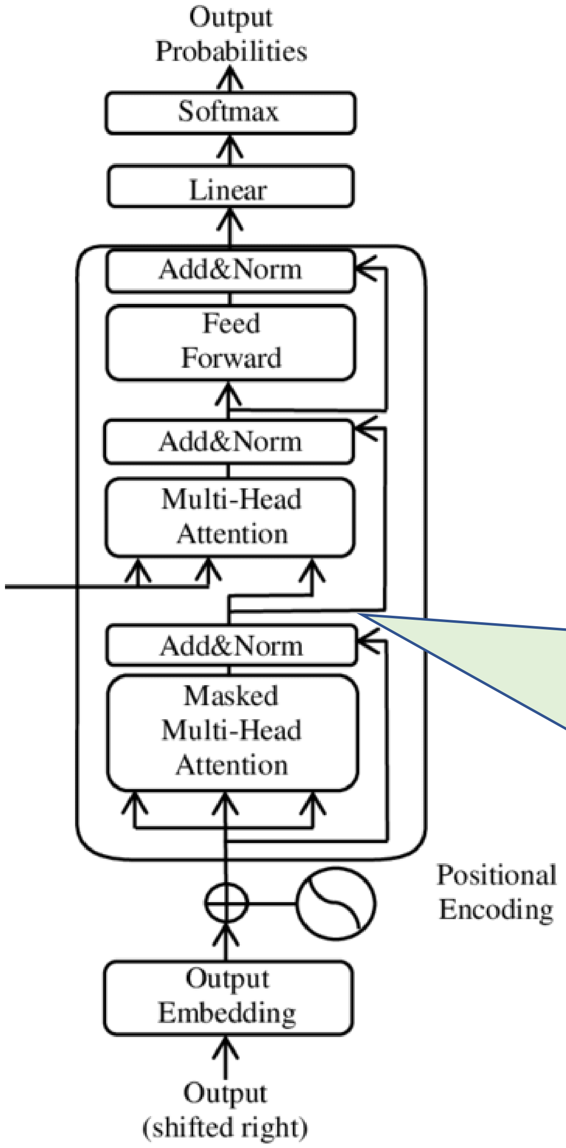
Multiple Heads: Each word (e.g., "Bootcamp") has multiple Q, K, V sets, with each head focusing on different context:
Head 1: "Leadership" with "AIware"
Head 2: "Bootcamp" with "is“

Masked Attention: When processing "is," only "AIware," "Leadership," and "Bootcamp" are visible; "awesome" is masked.

Combining Heads: Head outputs are concatenated, creating a rich final representation for each token.

Training a transformer - How do Decoder-only Transformers work?

Fifth, Stabilize and the residual information



Purpose: Stabilizes learning by combining outputs and normalizing them, helping the model handle complex relationships.

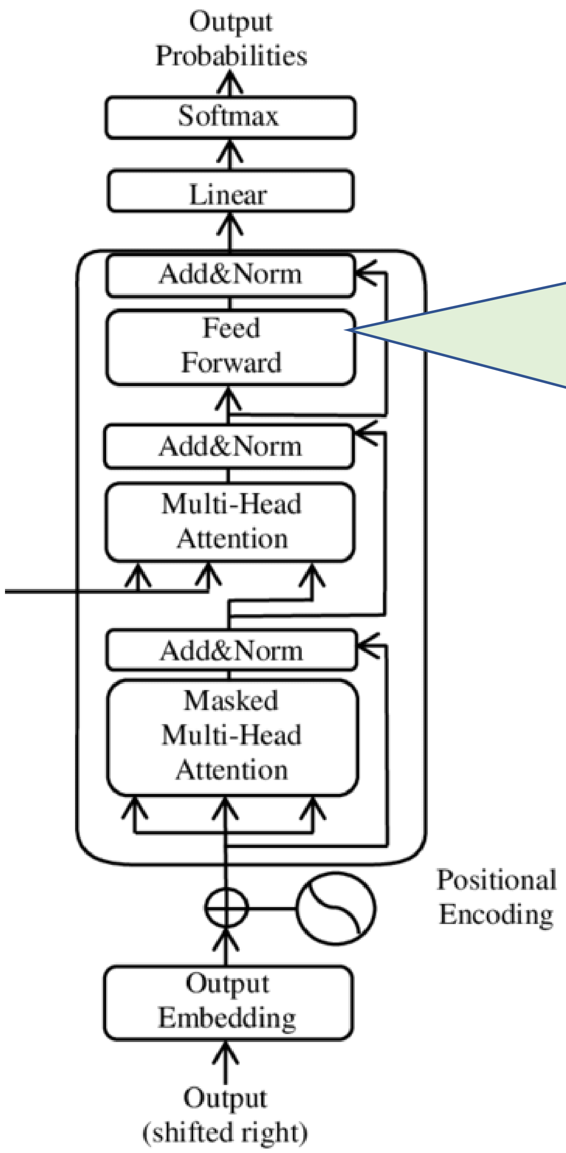
Addition: Adds the original input (e.g., embeddings for "AIware Leadership Bootcamp is") to the output of the self-attention layer, preserving the original information.

Normalization: Applies layer normalization to the summed result, ensuring consistent scale and improving training stability.

Result: Produces a normalized output that combines both original input and new contextual information, ready for the next layer.

Training a transformer - How do Decoder-only Transformers work?

Sixth, Enrich the representation of each token



Purpose: Transforms each token's representation independently, adding depth and flexibility to the model's understanding.

Two Linear Layers: Applies two transformations with a ReLU activation in between, enhancing the representation's complexity.

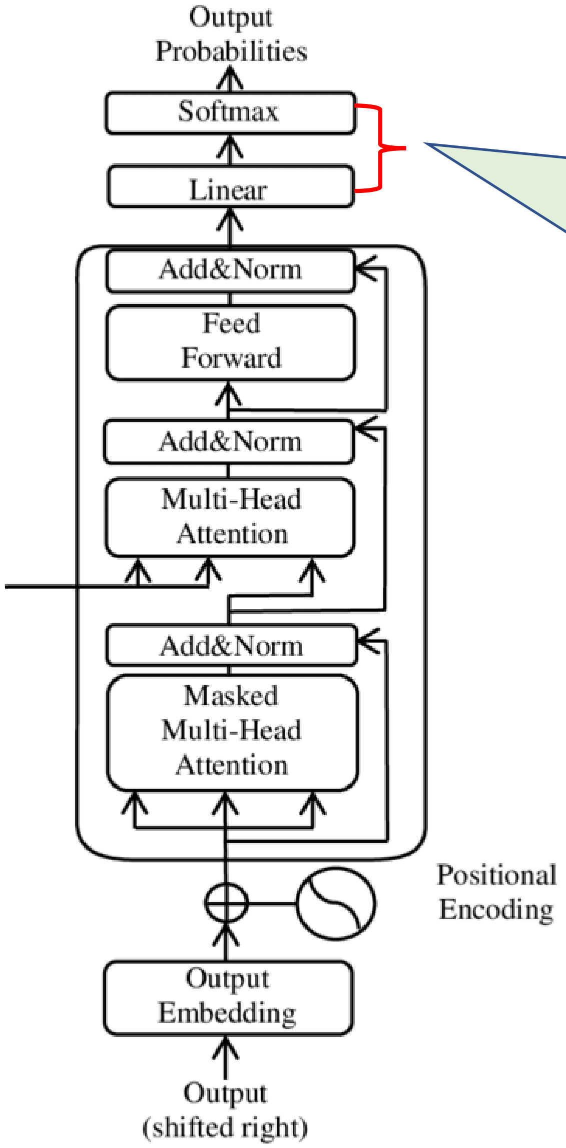
Example: For "AIware," transforms its embedding to capture more nuanced features.

Independent Processing: Each token (e.g., "AIware," "Leadership") passes through the feed-forward layer independently, focusing on enriching individual token information.

Output: Returns an updated, enriched representation for each token, now ready for the next attention layer.

Training a transformer - How do Decoder-only Transformers work?

Seventh, Generate output probabilities



Purpose: Converts each token's final representation into a probability distribution over the vocabulary to predict the next word.

Linear Layer: Projects each token's output (e.g., "is") to match the vocabulary size, creating scores for every possible next word.

Example: Outputs scores for words like "awesome," "great," "challenging," etc.

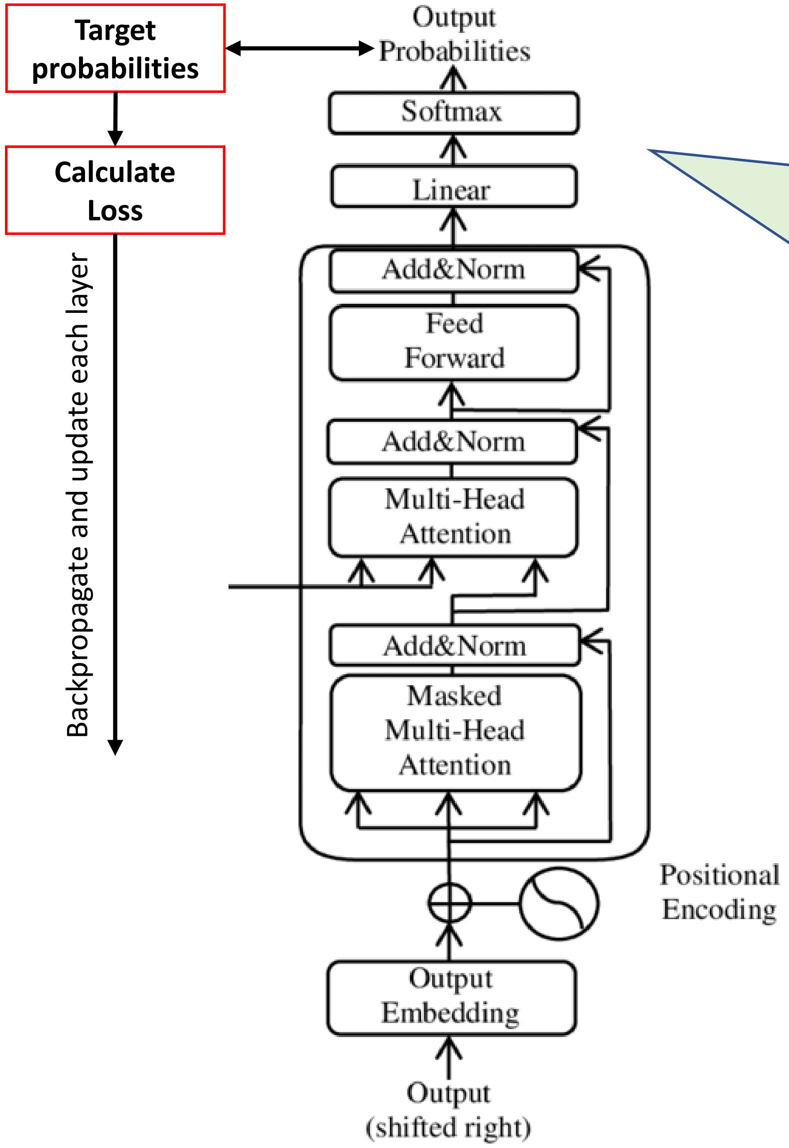
Softmax Layer: Applies softmax to these scores, turning them into probabilities that sum to 1.

Example: "awesome" might have the highest probability, signaling it as the likely next word.

Result: The model selects the word with the highest probability as the next token, completing the prediction.

Training a transformer - How do Decoder-only Transformers work?

Eighth, Teach the model to do better



Purpose: Adjusts model weights to minimize prediction error, improving accuracy over time.

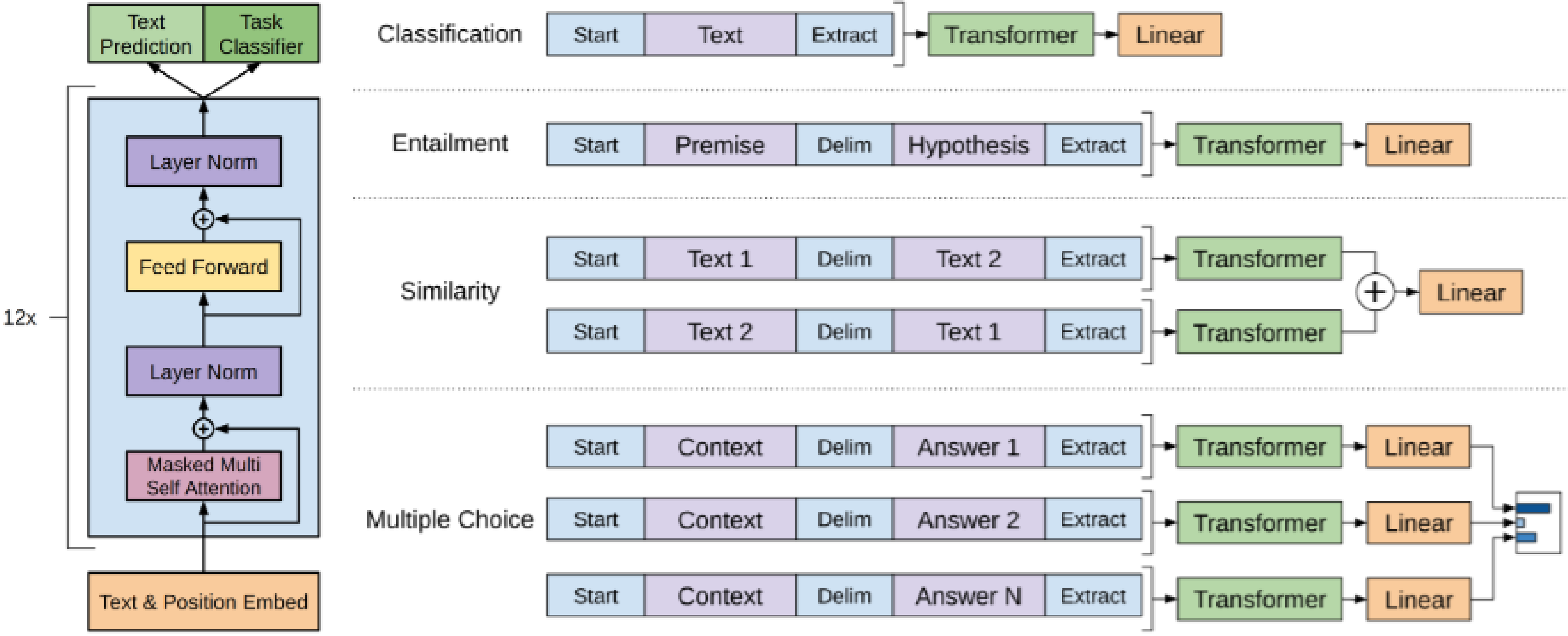
Calculate Loss: Compares predicted output (e.g., probability for "awesome") to the actual target. If "awesome" has low probability, loss is high.

Backpropagation: The model calculates gradients, which show how much each weight contributed to the error. Gradients are computed for each layer, starting from the output and moving backward through the model (from softmax to self-attention layers).

Weight Update: Weights (in embedding, attention, and feed-forward layers) are adjusted using an optimization algorithm (like Adam) to reduce loss.

This update process gradually improves the model's predictions.

Early GPT Transformer architecture and training

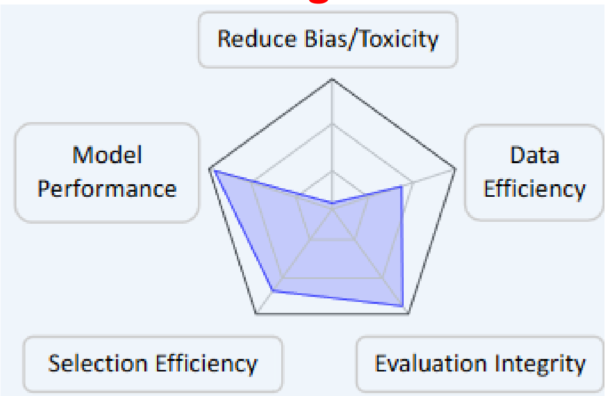


Radford et al., Improving Language Understanding by Generative Pre-Training

But, wait! What about the data?

Raw data, typically in large volumes (order of Terabytes)

Targets



Houston, we have an FM!

Albalak et al., A Survey on Data Selection for Language Models

Used Datasets for training FMs

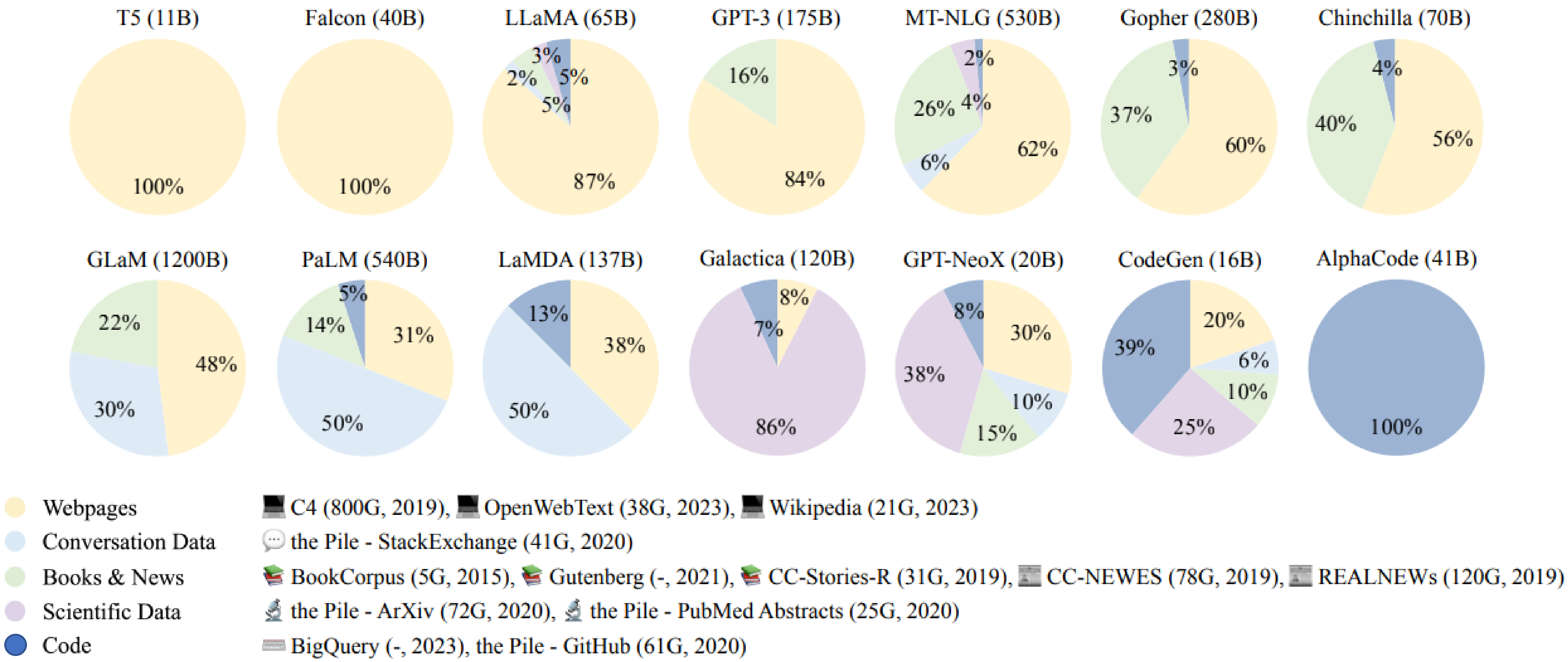


Fig. 5: Ratios of various data sources in the pre-training data for existing LLMs.

Zhao et al., A Survey of Large Language Models

Overview of the session

- ❑ **A brief intro to pre-training Foundation Models (FM):** An introduction to how a FM is pre-trained
- ❑ **Why do we have to align FMs:** Motivating the need for Alignment
- ❑ **Taxonomy of Alignment Engineering**
  - ❑ Data Alignment
  - ❑ Model Alignment
    - ❑ Finetuning
    - ❑ Online alignment
    - ❑ Offline alignment
  - ❑ Alignment Evaluation
- ❑ **Curriculum Learning**

Pretrained (FMs) Isn't Practical: Alignment for Real-World Use

Pre-trained FMs

Aligned FMs

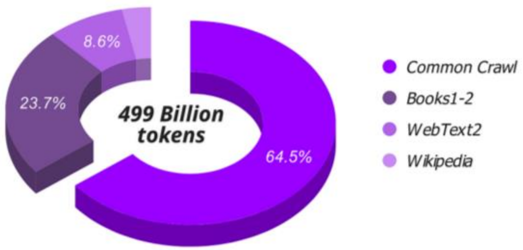
Alignment

The process of adjusting and guiding a FM's behavior through fine-tuning, prompt engineering, reinforcement learning from human feedback (RLHF), and other methods to ensure **it meets specific objectives, values, and safety standards for practical use.**

Question: Is ChatGPT a Foundation Model?

ChatGPT

GPT-3 is aligned to follow instructions to make it ChatGPT



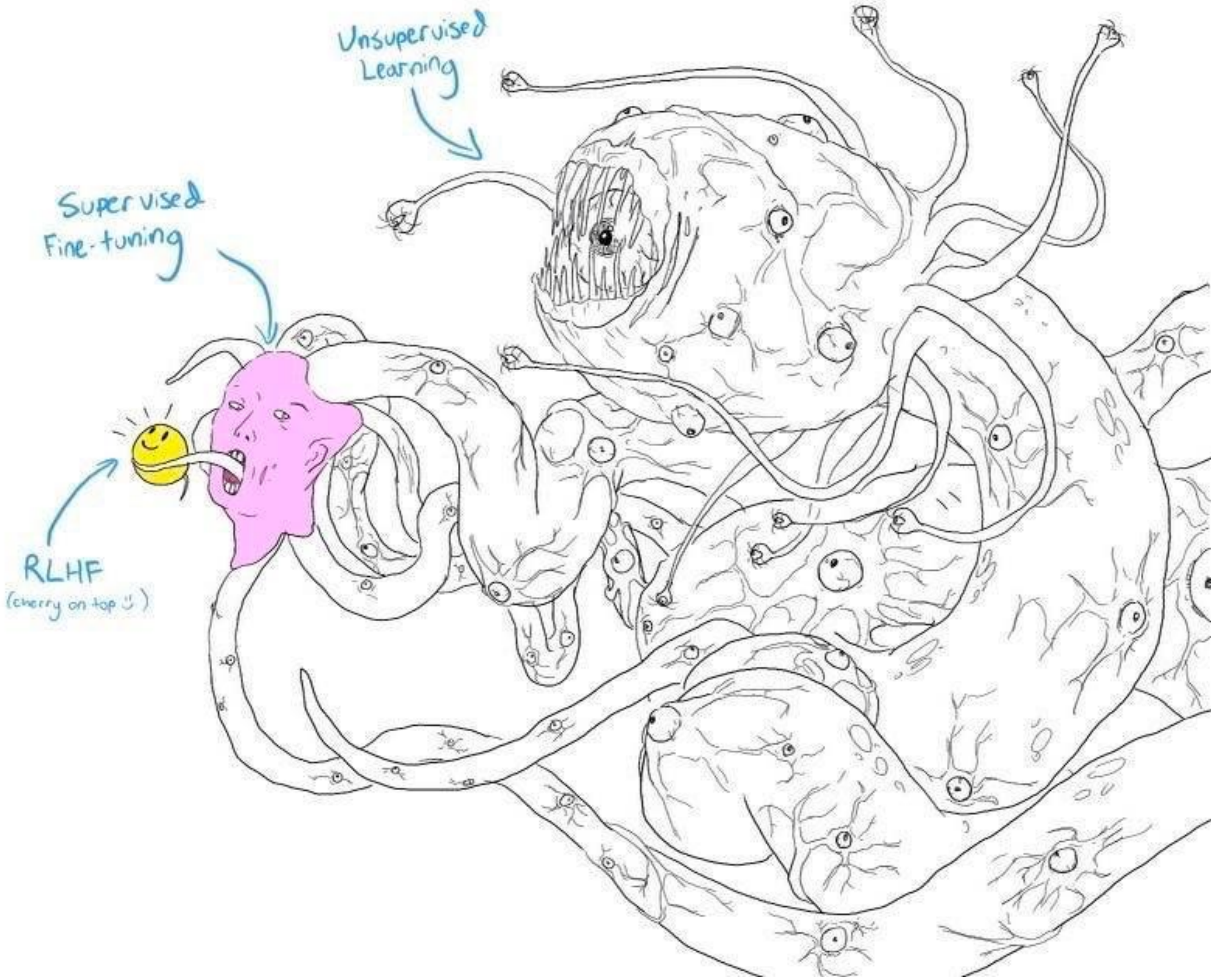
Pre-training

GPT-3

Alignment Engineering

ChatGPT

Alignment – A funny perspective

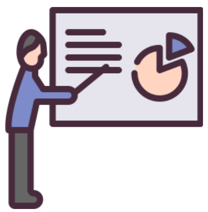


https://lastweekin.ai/p/dont-plan-for-agi-yeta

Ingredients required for Alignment Engineering

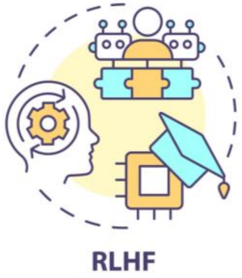
Pre-trained FM

Labeled data



Instructions

Preference



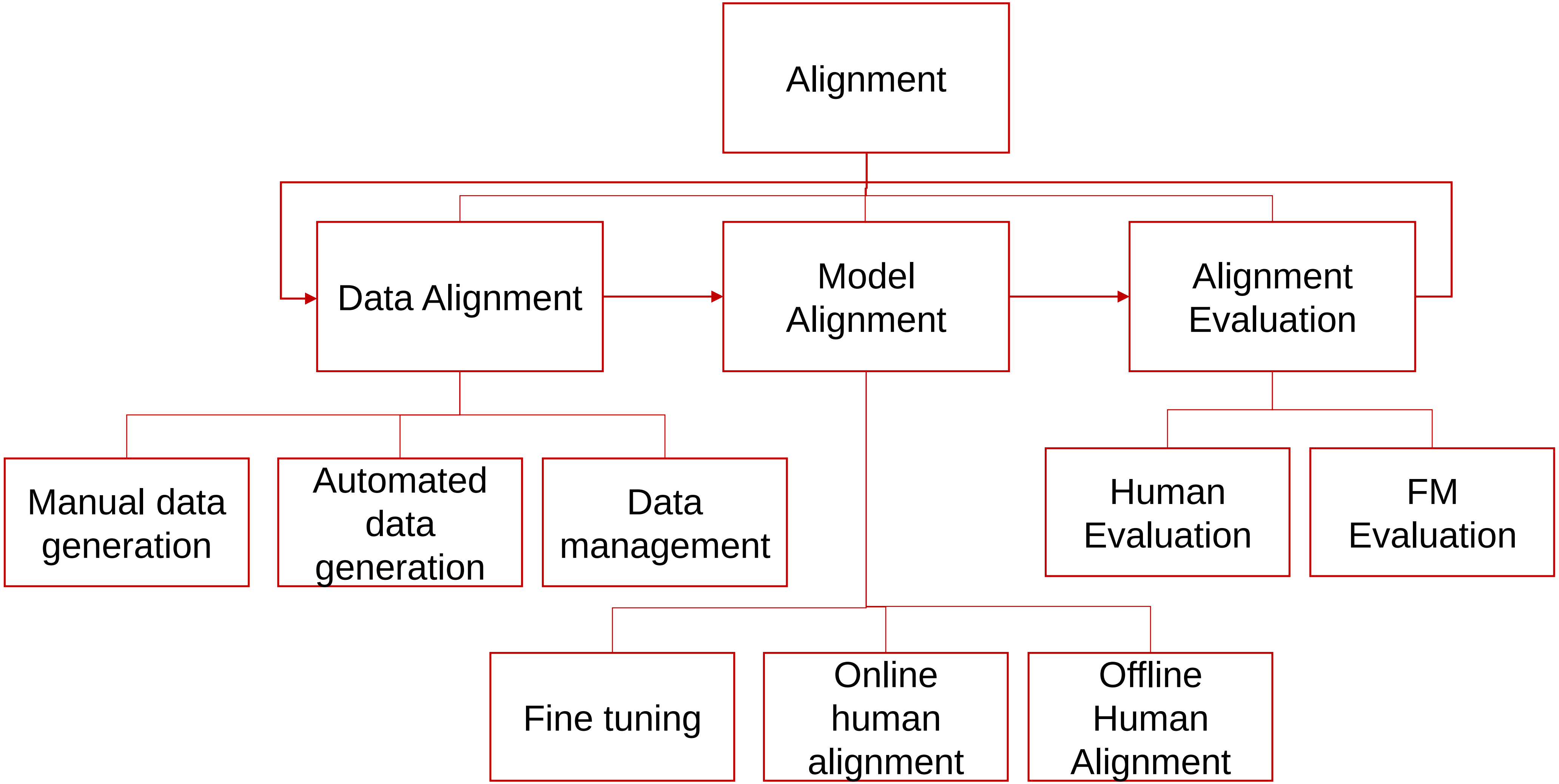
Data to teach the FM

Method to teach the FM

Overview of the session

- **A brief intro to pre-training Foundation Models (FM):** An introduction to how a FM is pre-trained
- **Why do we have to align FMs:** Motivating the need for Alignment
- **Taxonomy of Alignment Engineering**
  - Data Alignment
  - Model Alignment
    - Finetuning
    - Online alignment
    - Offline alignment
  - Alignment Evaluation
- **Curriculum Learning**

Taxonomy of Alignment Engineering



Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

Alignment Engineering: from coding to alignment

Example of Rain-Sensing Windshield Wipers

Codeware

Optical/Infrared/Acoustic Sensors
+
Lots source code

Alware

AI Model + a camera
+ lots of "tagged" pictures
+
"ZERO" source code

Developer writes code

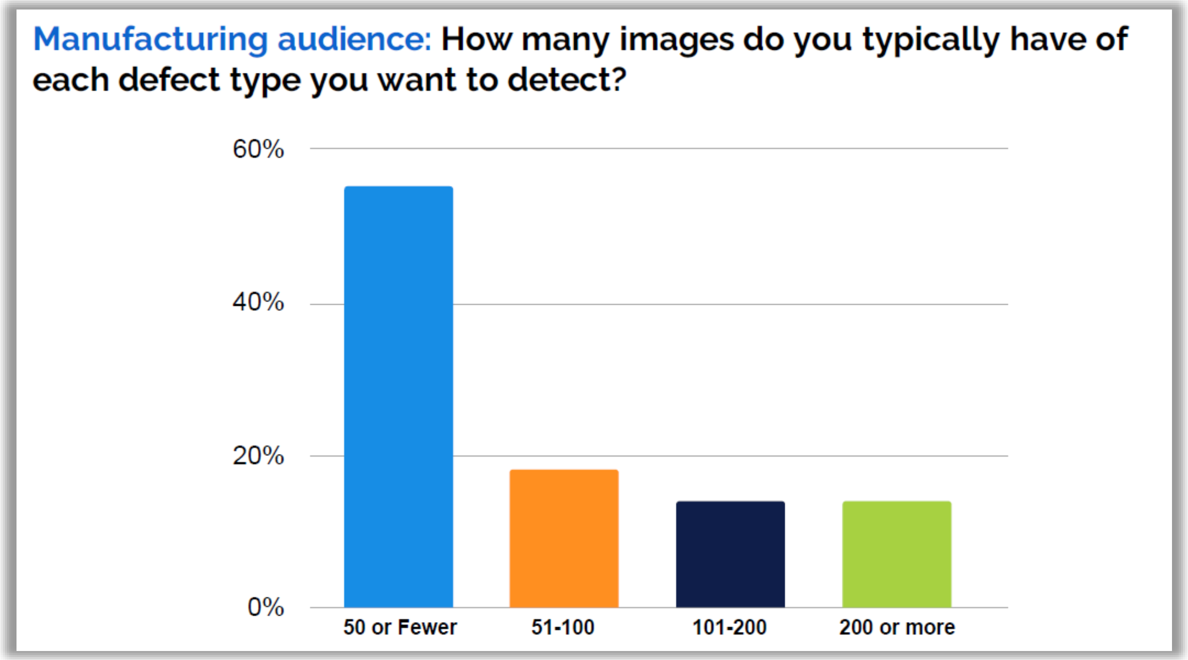
Fix a bug: "change code"

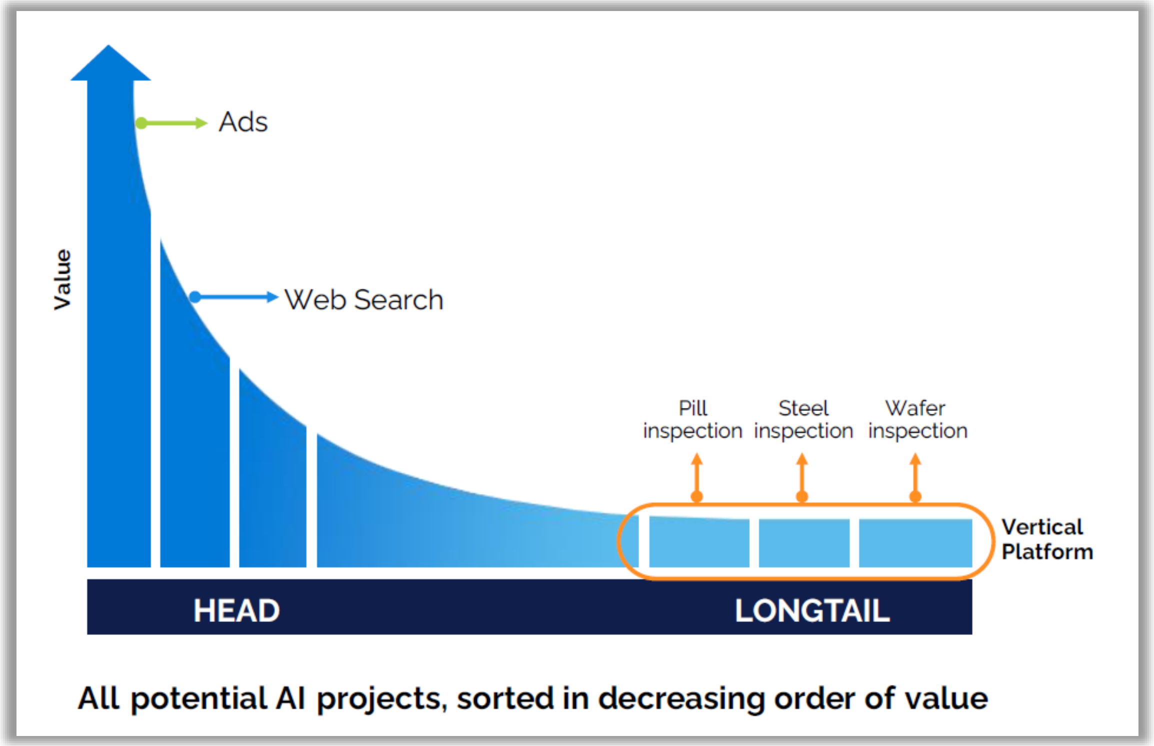
Developer defines a search space, data is the new code ← **Alignment Engineering**

Fix a bug: "take more pics"!!

Rajbahadur et al., Alware Leadership Bootcamp, Toronto, Canada, 2024

Alignment Engineering needs LOTS of "pricey" data



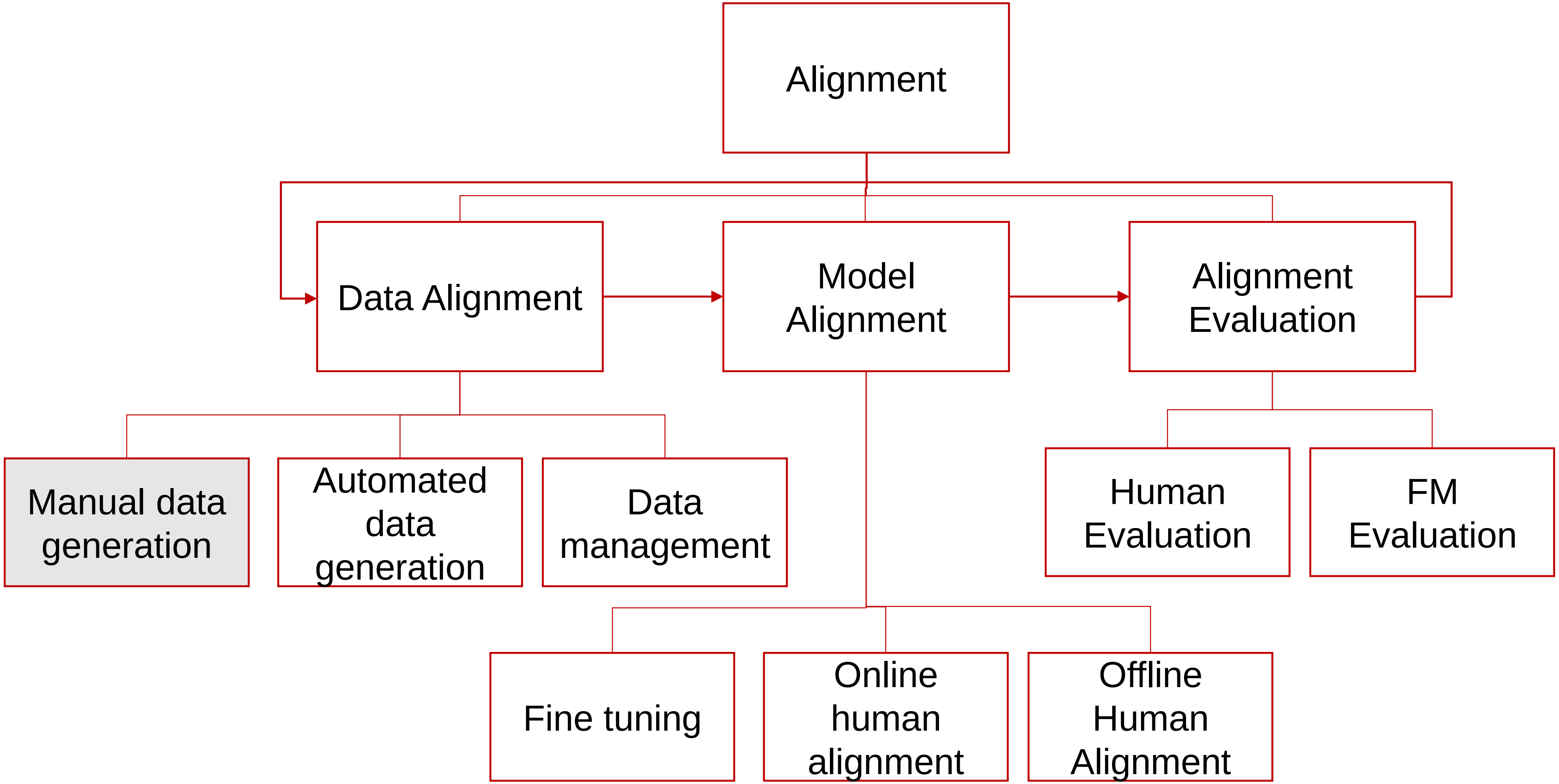


"tagged" data is rare/costly which led to limited success domains for AIware

Can we make tagging cheaper?

Can we develop AI models that require less data to learn?

Taxonomy of Alignment Engineering



Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

Scale AI’s approach to high-quality manual data labelling

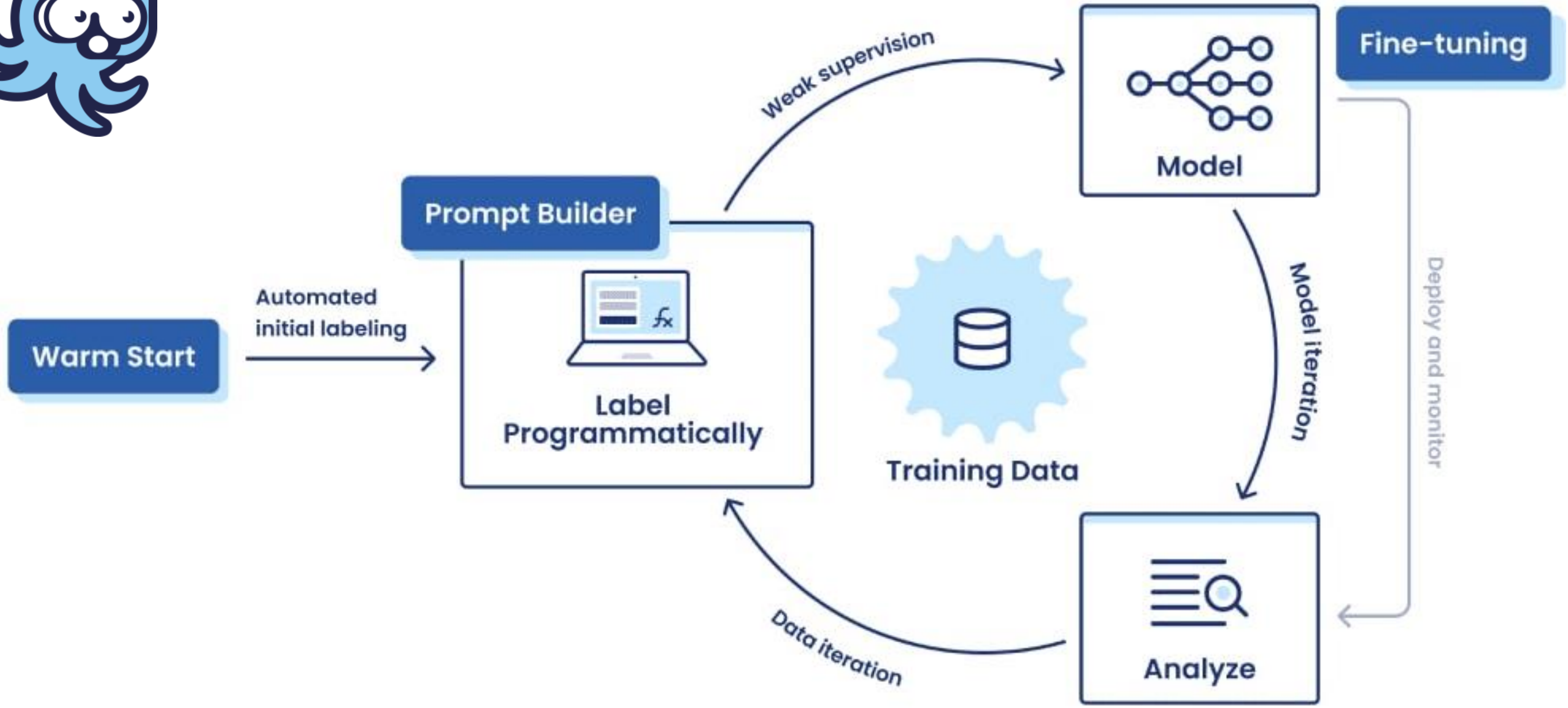
scale

OpenAI

Relies heavily on human labelers, which is very expensive and can be subject to bias.

https://scale.com/blog/how-to-label-1m-data-points-week

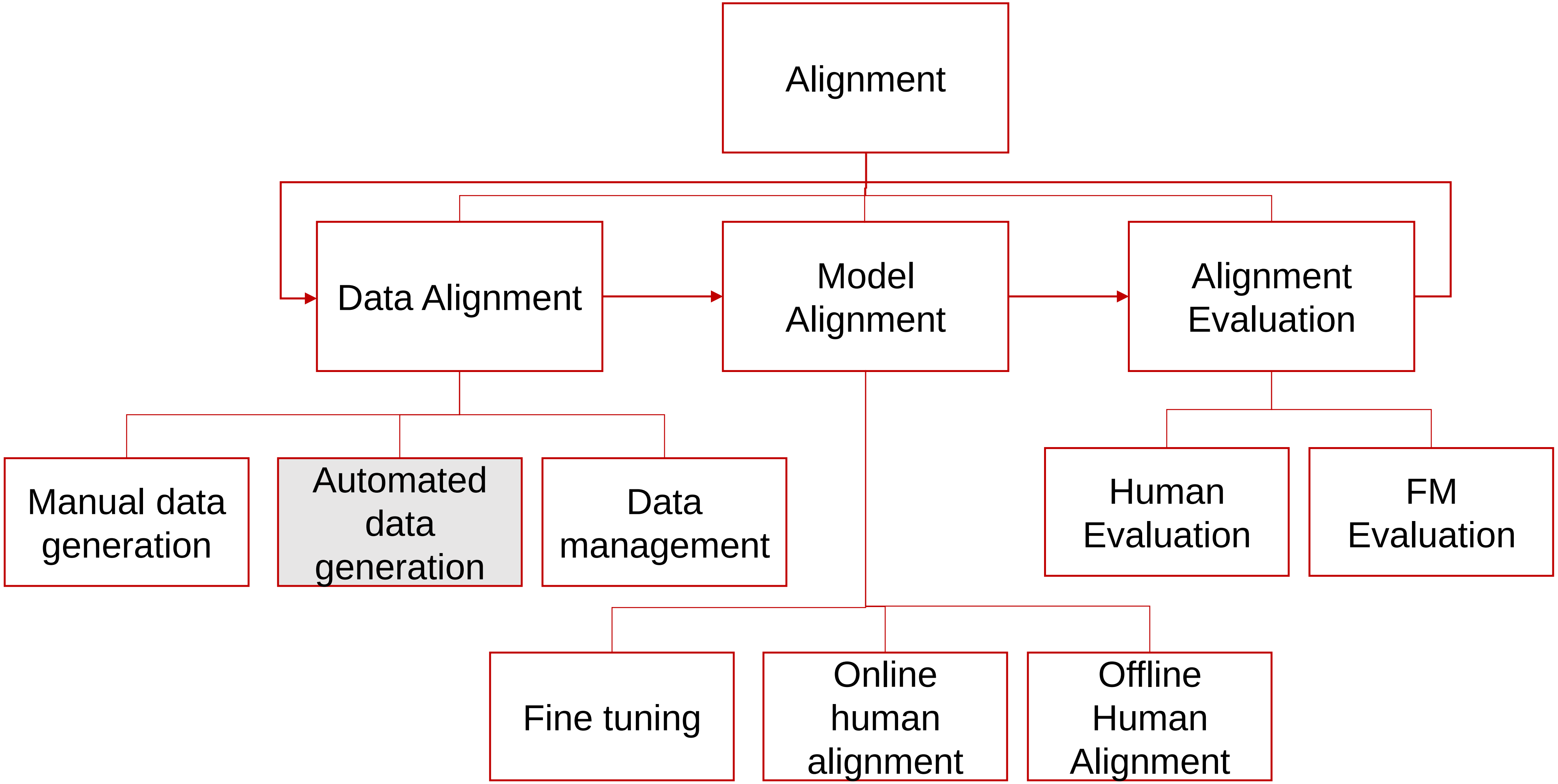
Data programming to semi-automated data generation



Requires expert intervention to iteratively refine the labelling functions which is costly.

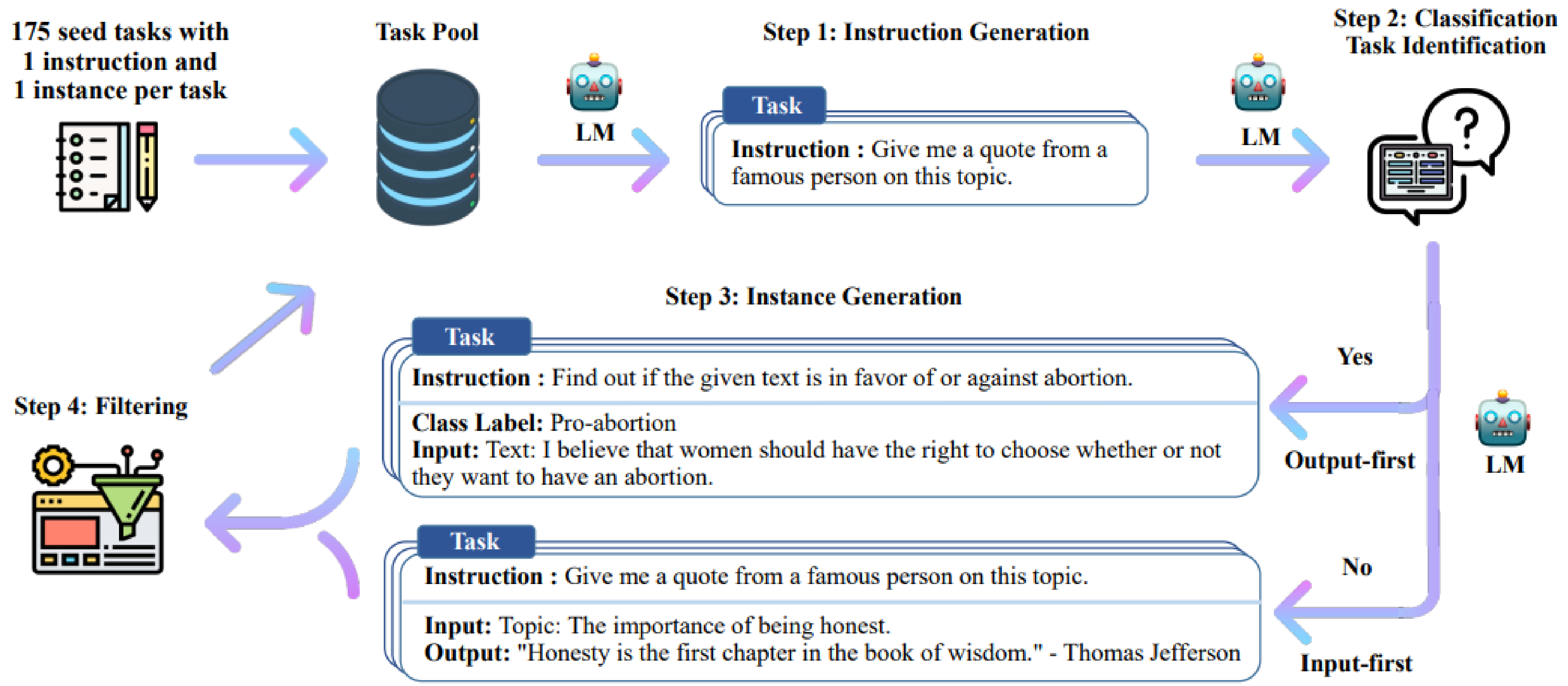
https://snorkel.ai/

Taxonomy of Alignment Engineering



Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

Self-Instruct – Leveraging FMs to synthetically generate data

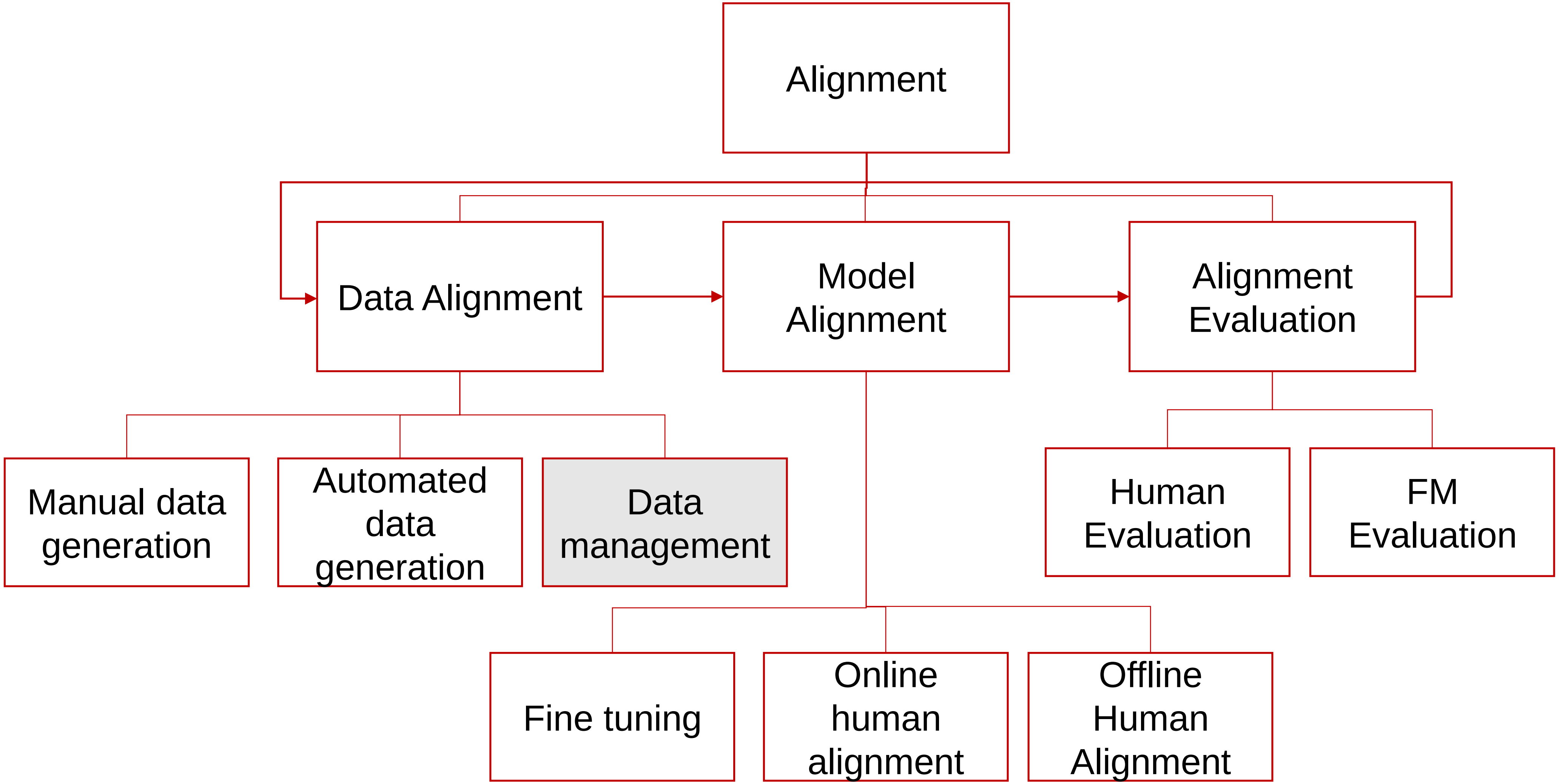


- Use FMs to generate data
- Start with a seed pool of prompts and then iteratively generate and refine task-specific data

Ensuring information diversity and overcoming FM-bias is challenging

Wang et al., Self-Instruct: Aligning Language Models with Self-Generated Instructions

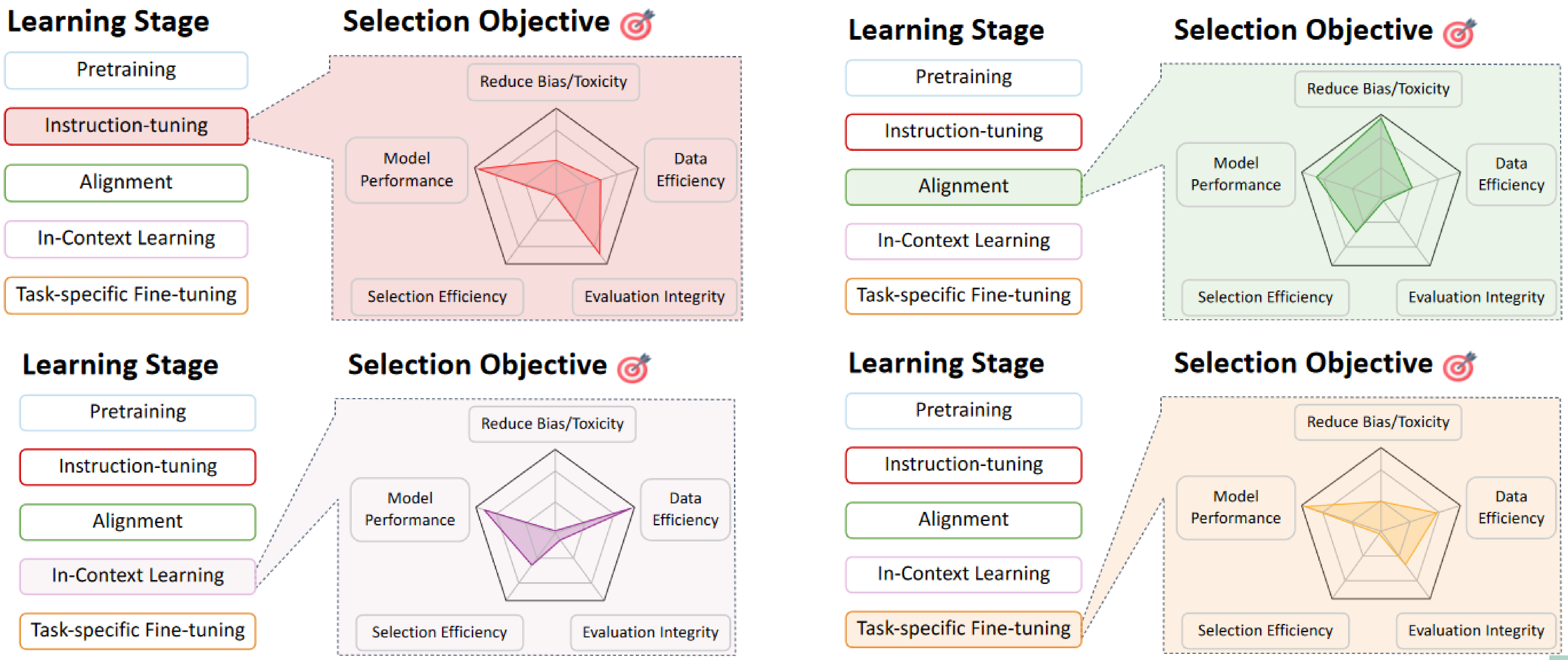
Taxonomy of Alignment Engineering



Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

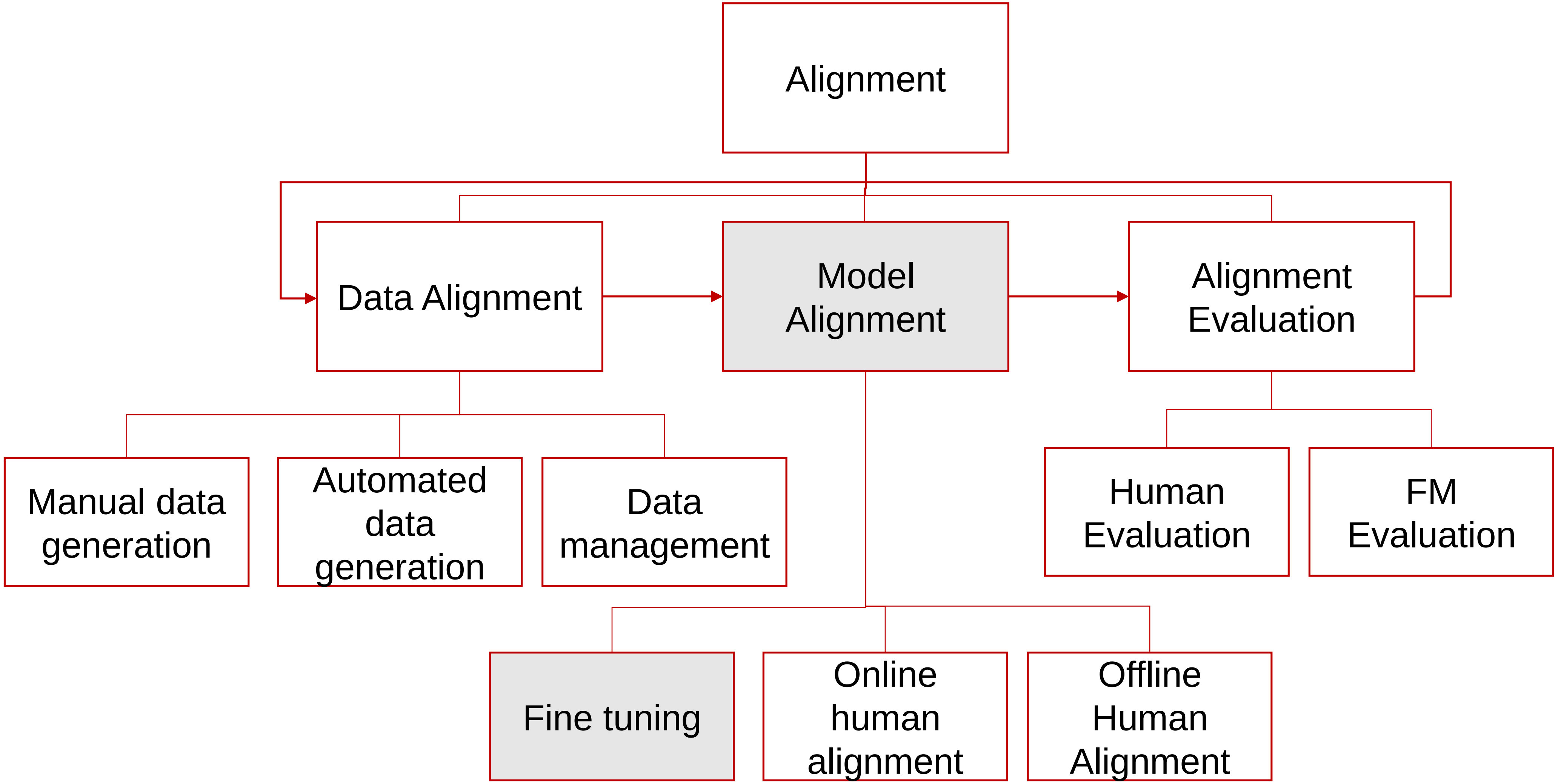
Alignment Data Management

Managing the quality, quantity, informativeness and diversity of the data



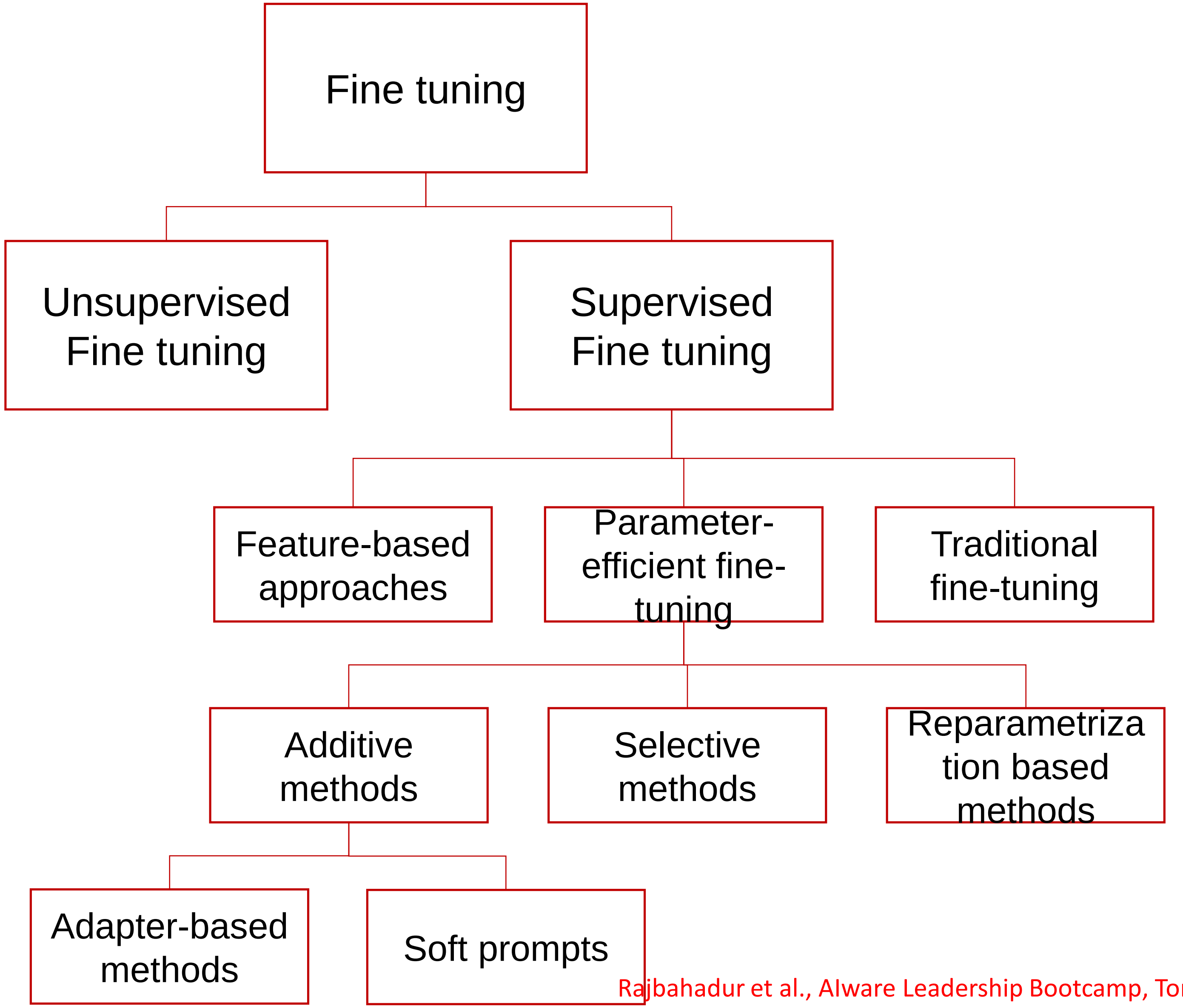
Albalak et al., A Survey on Data Selection for Language Models

Taxonomy of Alignment Engineering

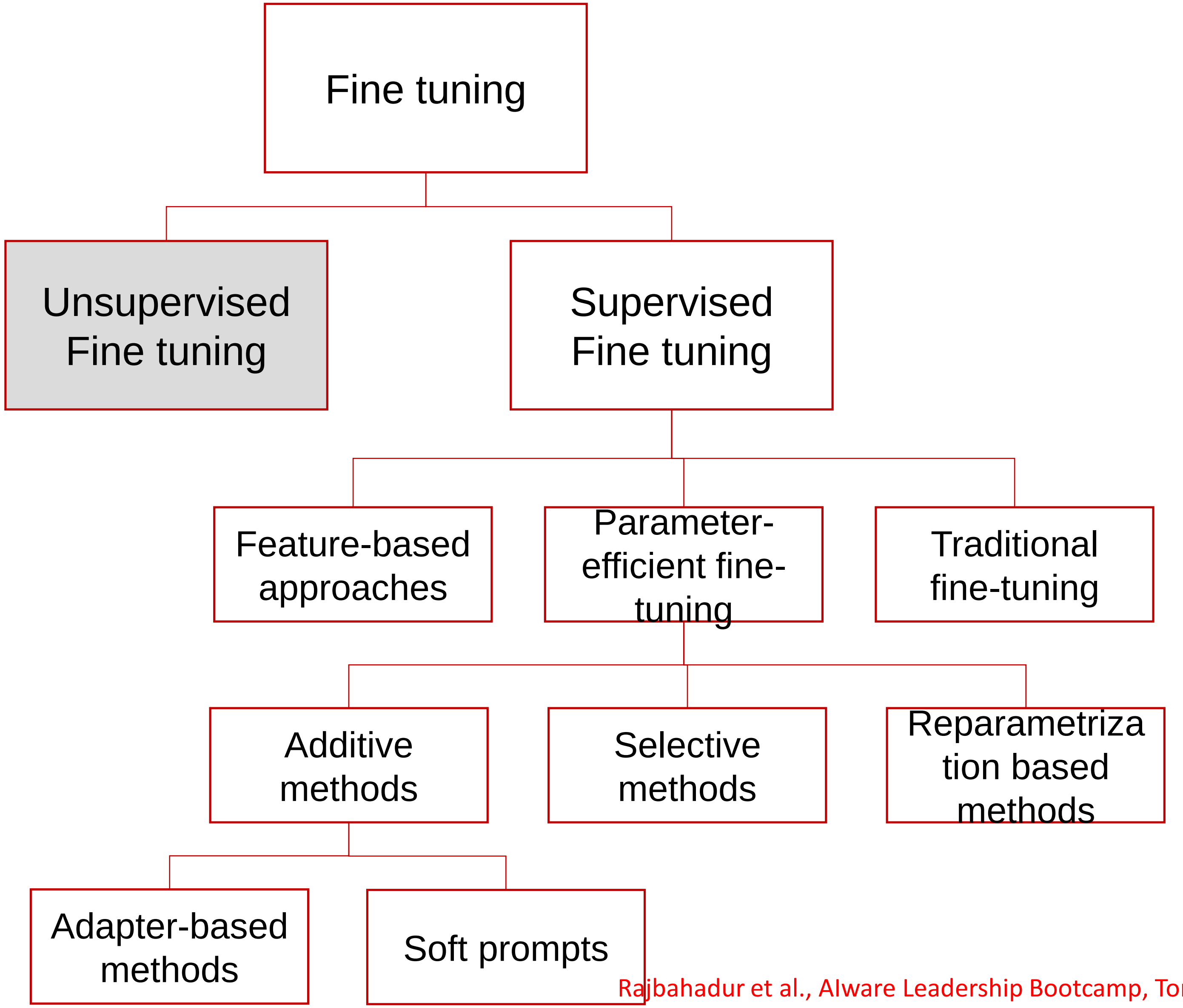


Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

Taxonomy of Fine-tuning approaches



Taxonomy of Fine-tuning approaches



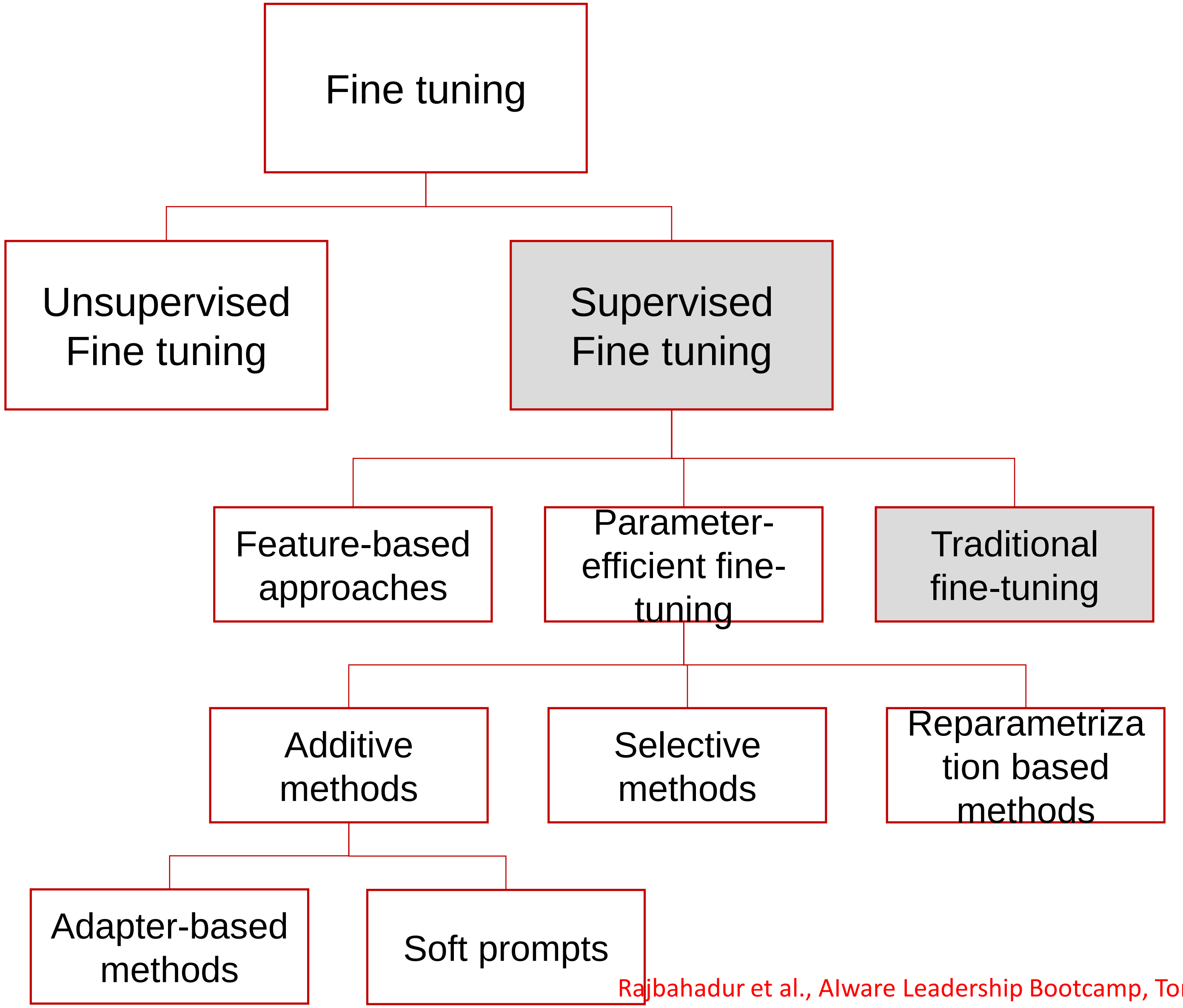
Fine-tuning approaches

Unsupervised Fine-tuning

- Unsupervised finetuning is a process in which a pretrained FM is finetuned on an unlabeled dataset from the target domain
- Typically a contrastive loss function is used to fine tune the model

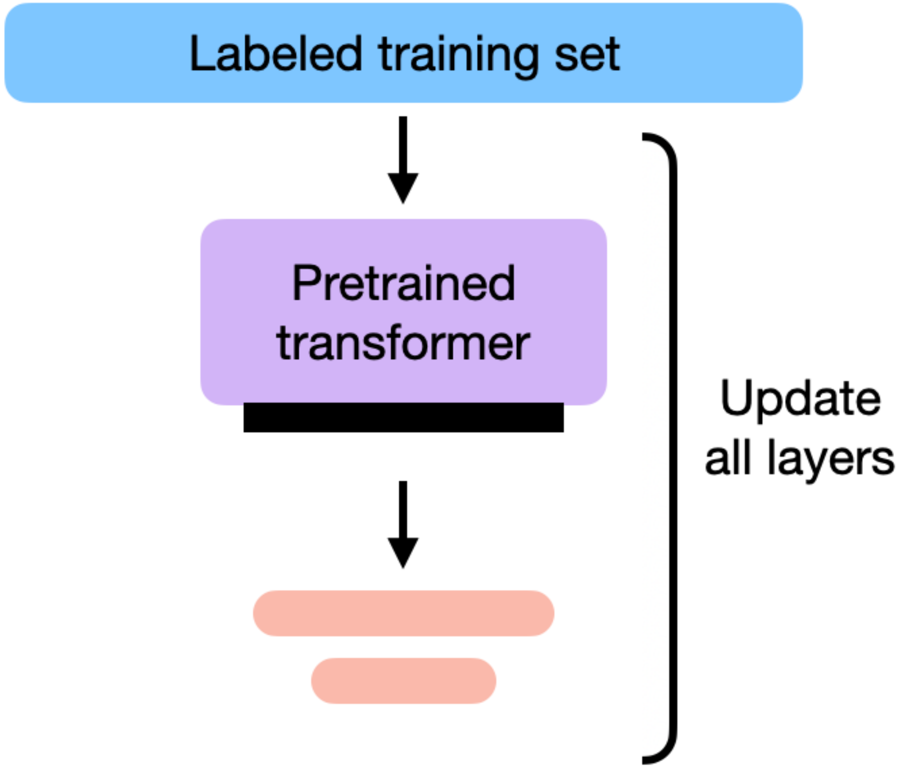
- Unsupervised fine-tuning typically requires a large amount of finetuning data
- Less computationally expensive, however, the results may not very good

Taxonomy of Fine-tuning approaches



Fine-tuning approaches

Supervised Fine-tuning

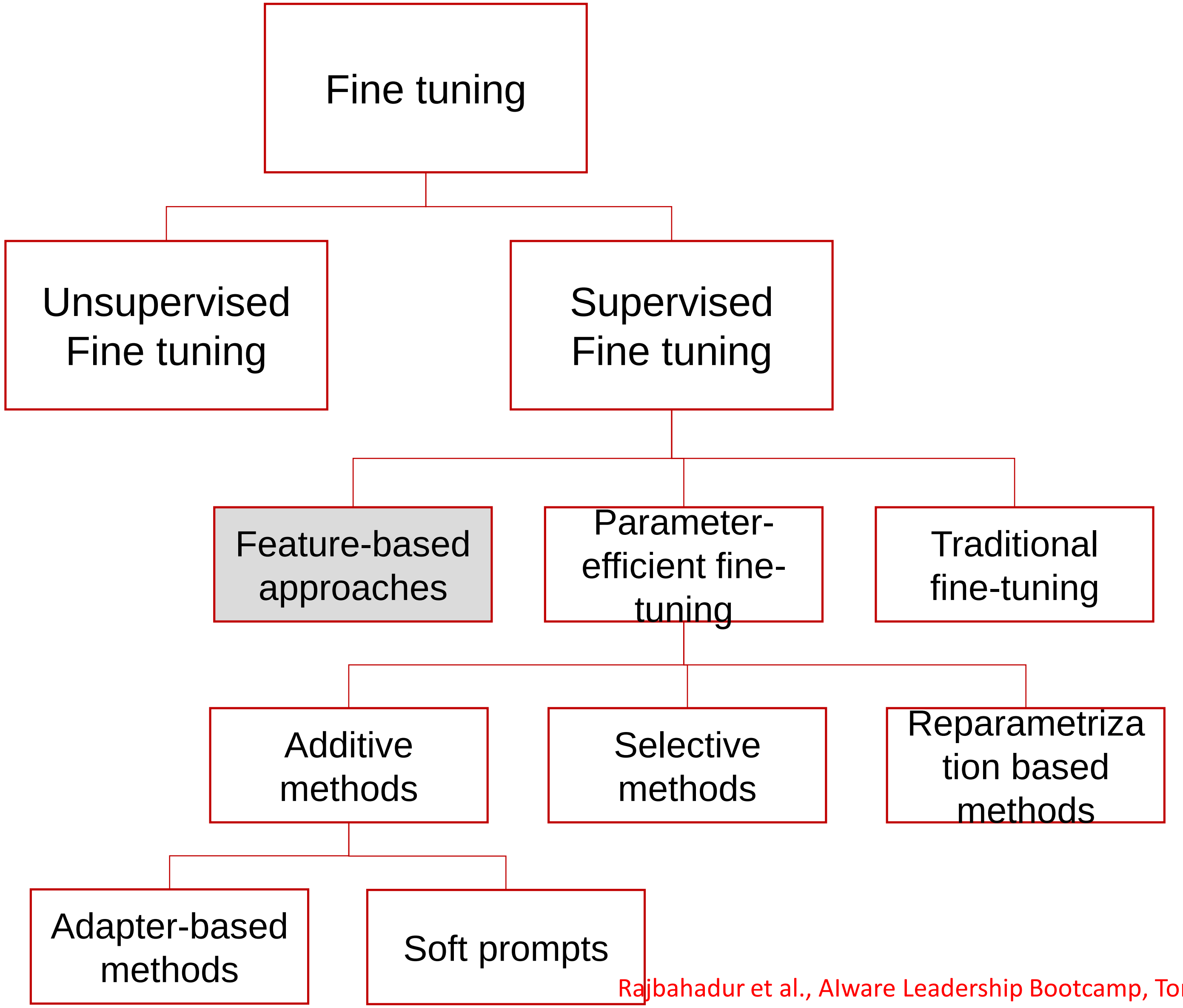


- Traditional fine tuning was used to build many of the state of the art FM models including GPT-3
- Involves in simply training the pretrained FM on task-specific labeled data

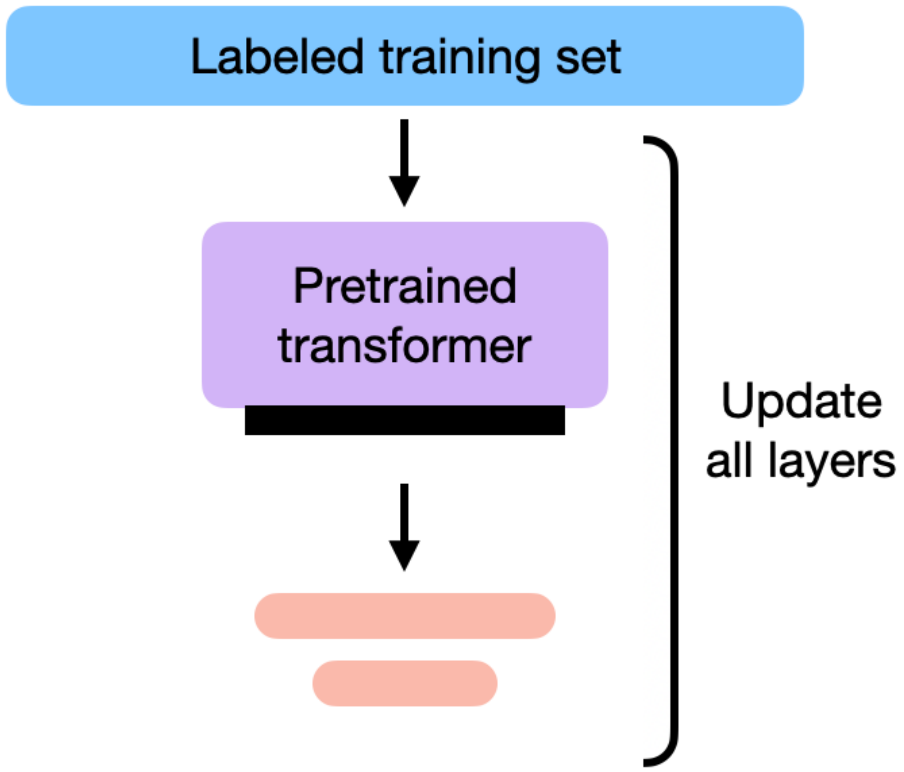
Very resource intensive. For instance, the vanilla fine-tuning of GPT-3 needs to update about 175,255 million parameters which is almost infeasible in both industry and academia.

https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Taxonomy of Fine-tuning approaches



Supervised Fine-tuning – Feature Based Approaches



- Feature based supervised fine tuning only uses the embeddings generated by the pre-trained FM and builds a classifier (or any other ML model that can use the embeddings)

The performance of the classifier depends on the classifier that is being used and its degrees of freedom. This approach is suitable only for classification tasks.

https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Taxonomy of Fine-tuning approaches

- Fine tuning
  - Unsupervised Fine tuning
  - Supervised Fine tuning
    - Feature-based approaches
    - Parameter-efficient fine-tuning
      - Additive methods
        - Adapter-based methods
        - Soft prompts
      - Selective methods
      - Reparametrization based methods
    - Traditional fine-tuning

Why Parameter Efficient Fine-tuning (PEFT)?

Traditional Supervised Fine-tuning is Expensive

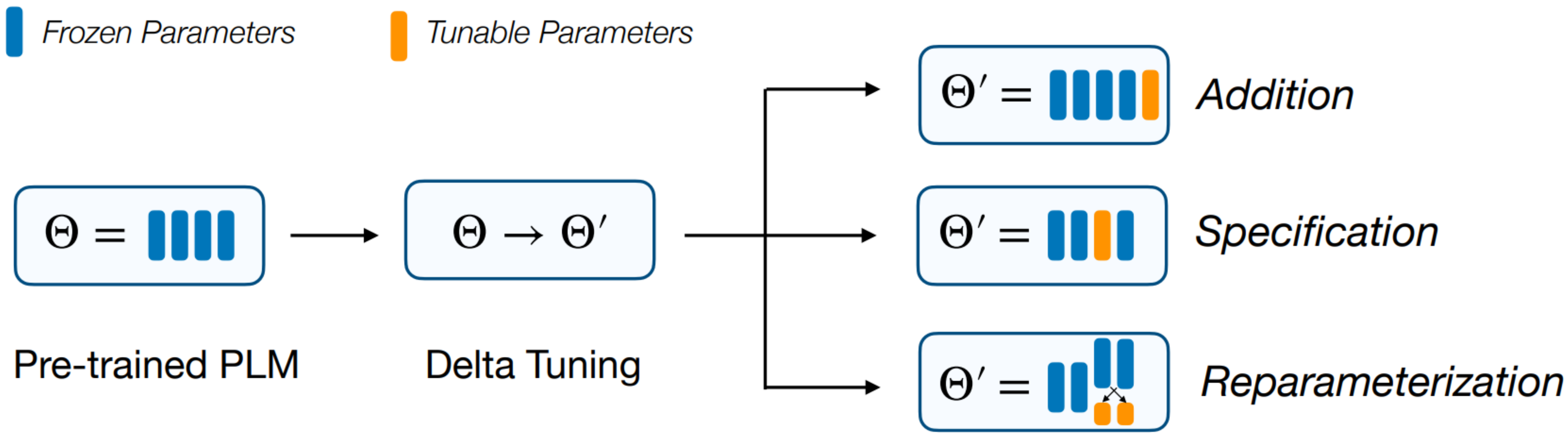
- To finetune BLOOM-176B, we would require almost 3TB of GPU memory which is approximately 72 80GB A100 GPU.
- In addition, one has to store all the weights, gradients and intermediate states which further runs up the cost.

PEFT to the Rescue!

- PEFT methods tackle this issue by training a small subset of existing or newly added model parameters, maintaining the FM's performance.
- LoRA, a PEFT method might need to train only 37.7 M parameters in place of 175 B parameters

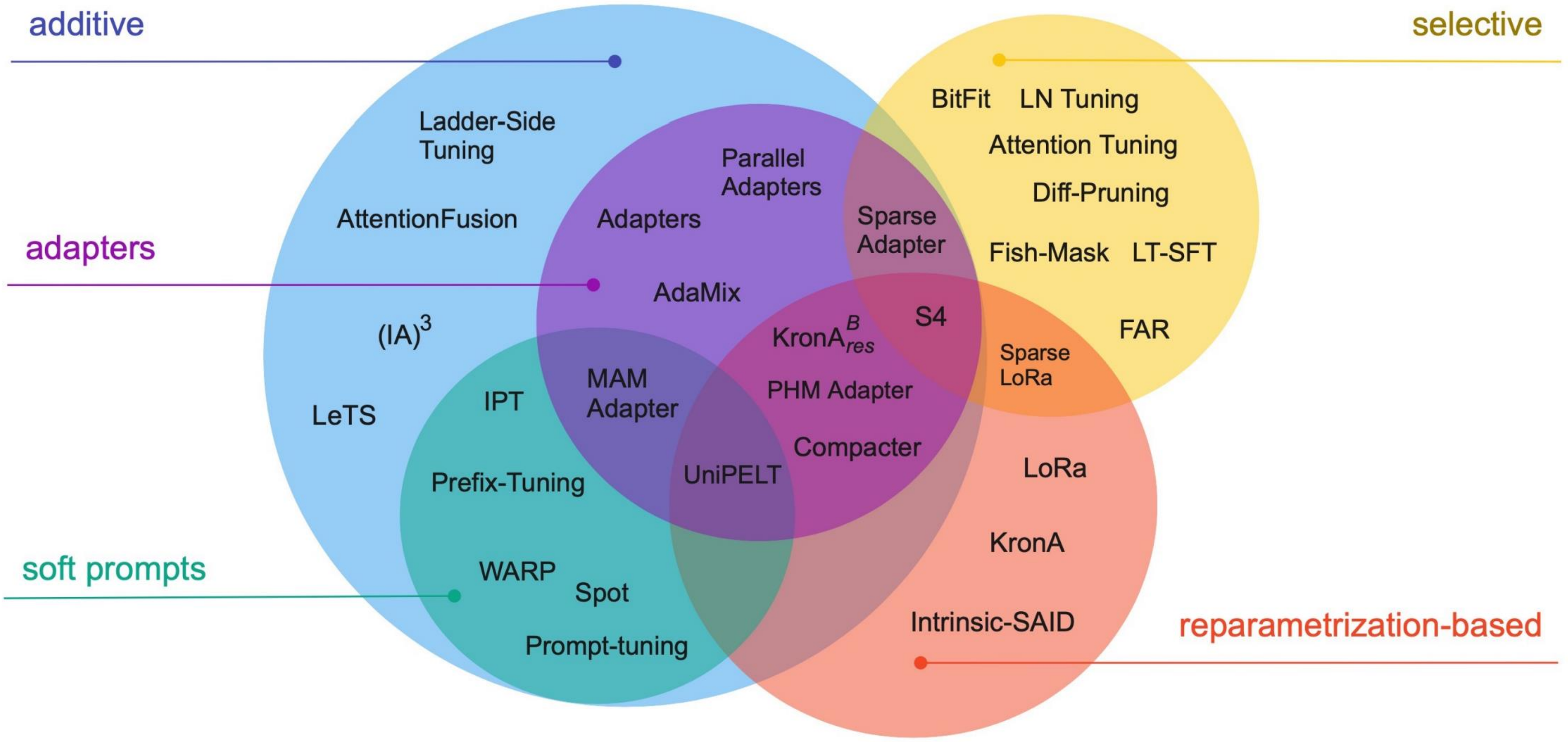
https://deepsense.ai/how-to-train-a-large-language-model-using-limited-hardware/

PEFT methods can be largely classified into three family of methods



Ding et al., Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models

PEFT methods can be largely classified into three family of methods



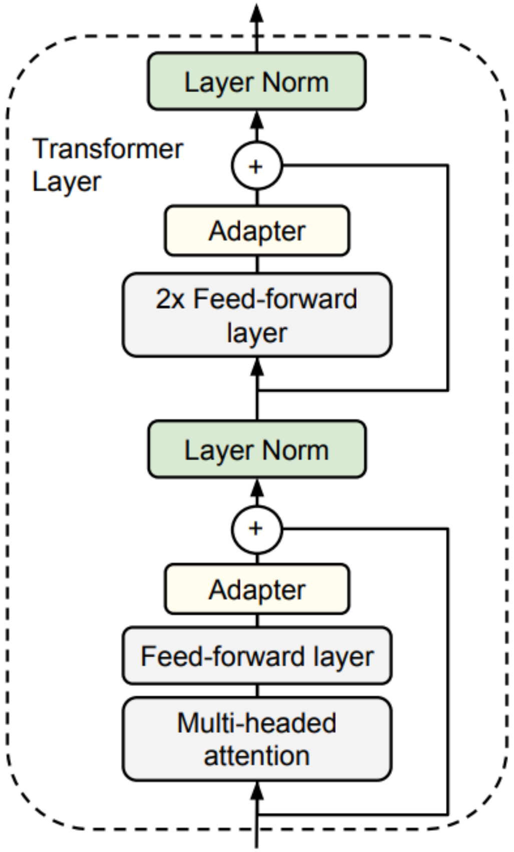
Lailin et al., Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning

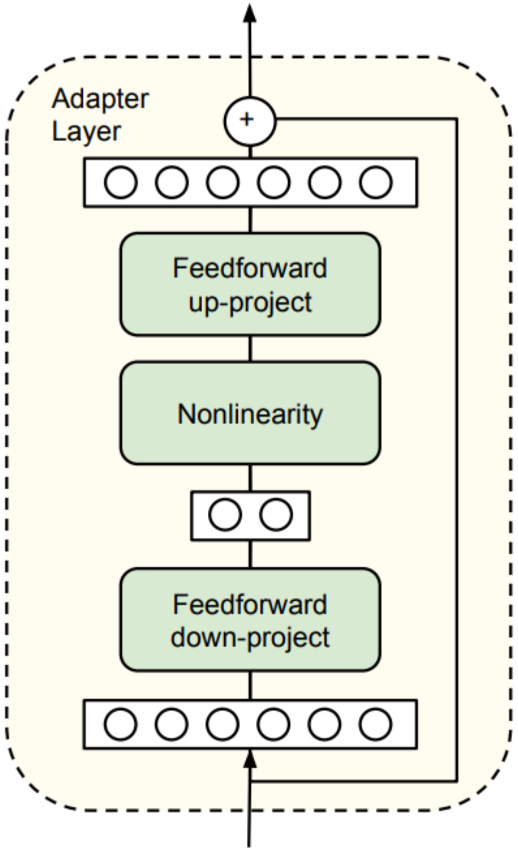
Taxonomy of Fine-tuning approaches

- Fine tuning
  - Unsupervised Fine tuning
  - Supervised Fine tuning
    - Feature-based approaches
    - Parameter-efficient fine-tuning
      - Additive methods
        - Adapter-based methods
        - Soft prompts
      - Selective methods
      - Reparametrization based methods
    - Traditional fine-tuning

Parameter Efficient Fine-tuning approaches

Adapters

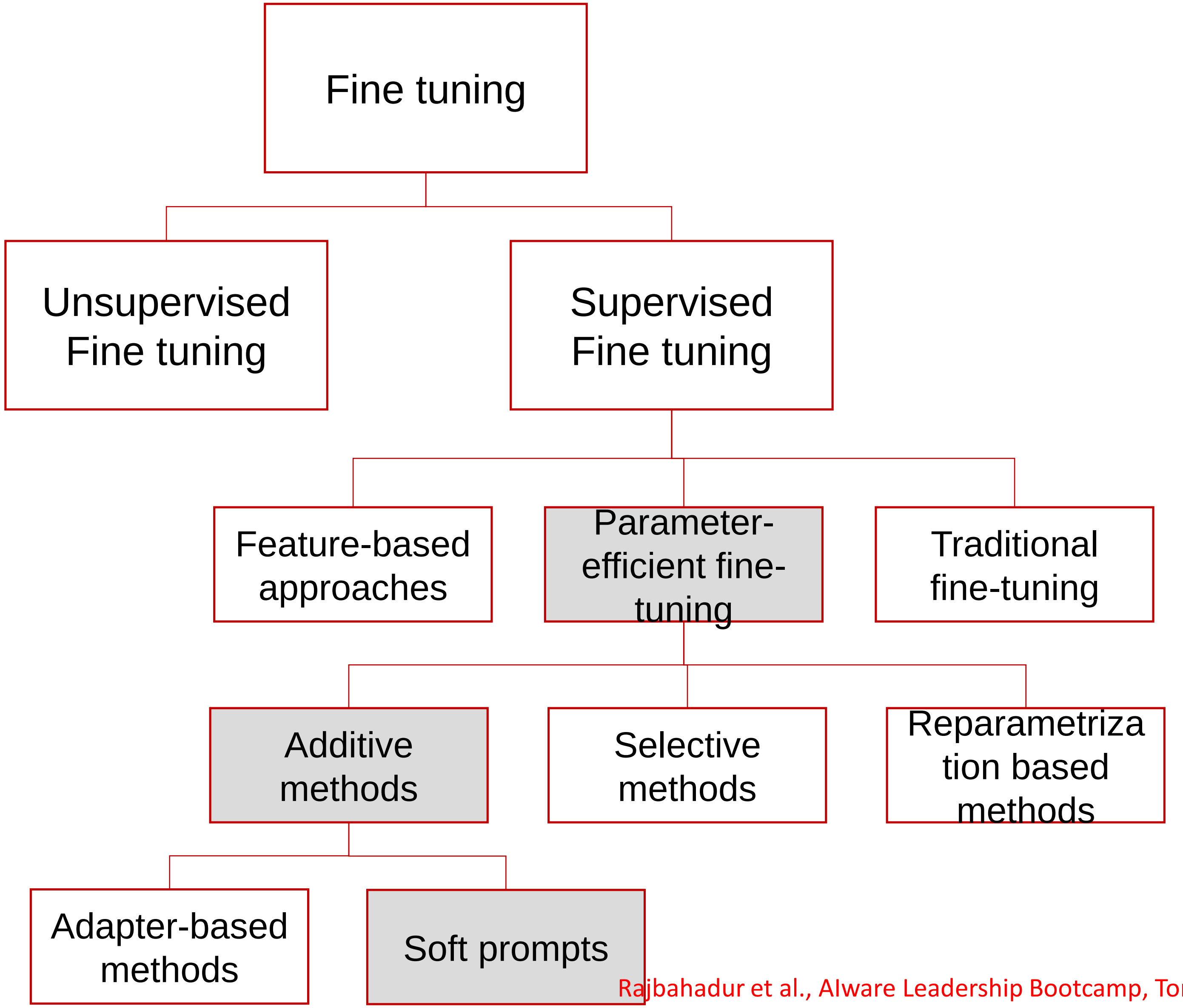




- The adapter family of methods typically adds adapter layers between the different transformer layers and only finetune these layers while keeping the weights of the other layers frozen.
- Therefore the number of trainable parameters become significantly less.
- These adapter layers are also task specific (e.g., medical assistant), so they can be transferred to other models with similar architecture (though non-trivially)

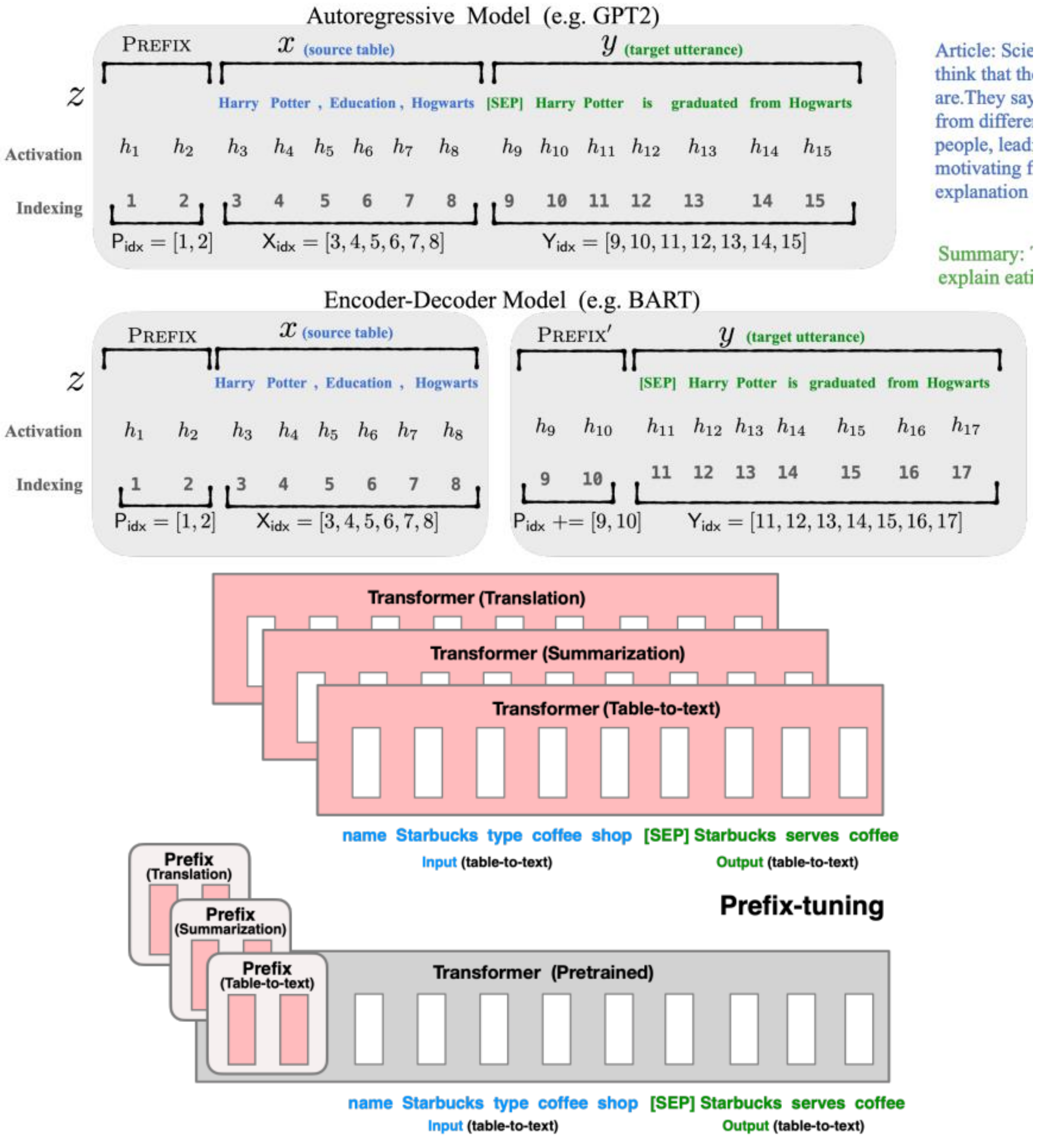
https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Taxonomy of Fine-tuning approaches



Parameter Efficient Fine-tuning approaches

Prompt and Prefix Tuning

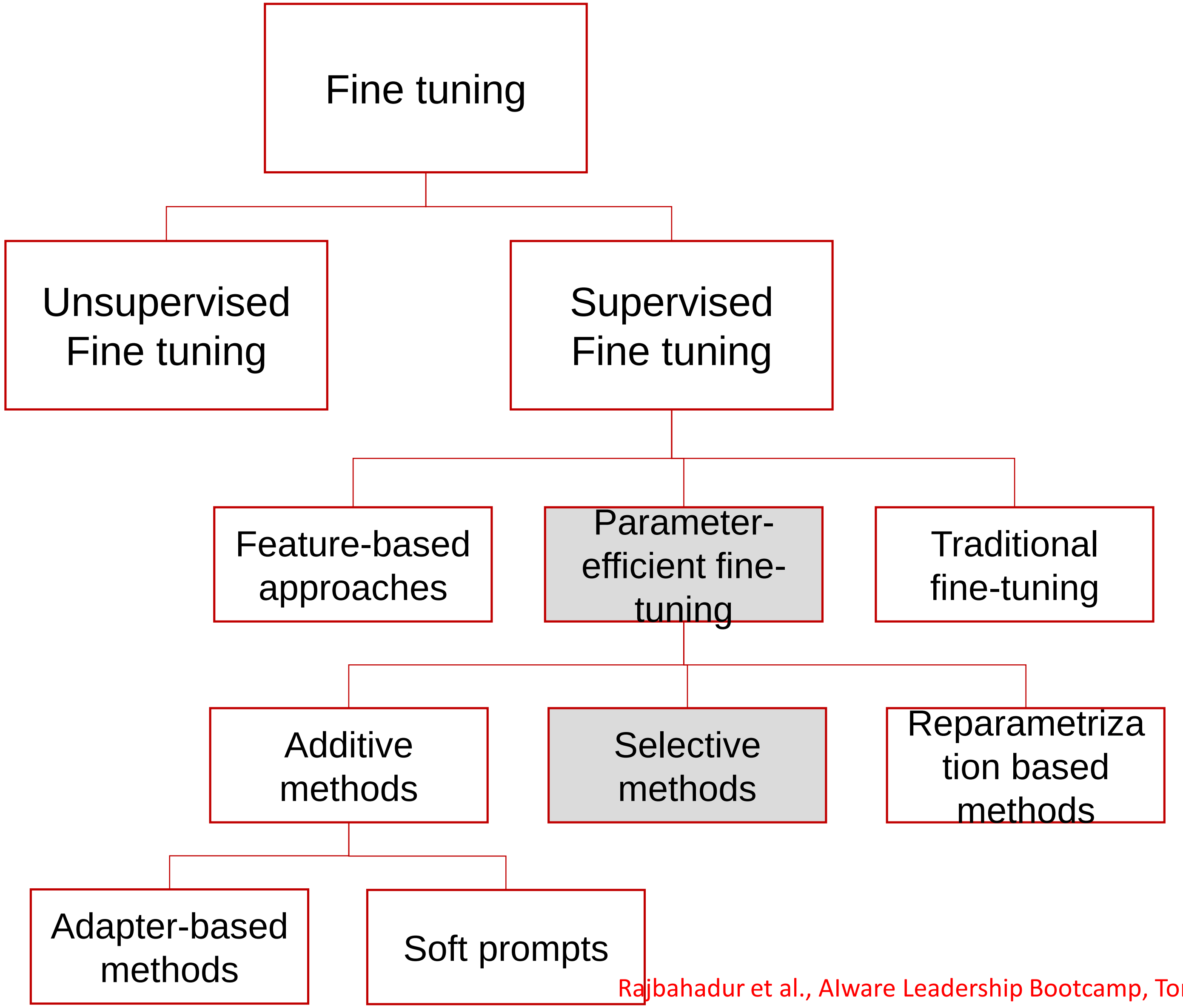


- **Prompt Tuning:** Prompt tuning optimizes a fixed number of task-specific prompt tokens, inserted at the start of the input. This approach steers the model's responses by learning prompts that prompt the frozen model toward the desired task output.

- **Prefix Tuning:** Prefix tuning involves adding trainable tokens, called prefixes, to the input of a frozen model. These prefixes act as context, guiding the model to generate task-specific responses without modifying the original model's parameters.

Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation; Lester et al., The Power of Scale for Parameter-Efficient Prompt Tuning

Taxonomy of Fine-tuning approaches



Parameter Efficient Fine-tuning approaches

BitFit

Concretely, the BERT encoder is composed of L layers, where each layer ℓ starts with M self-attention heads, where a self attention head (m, ℓ) has *key*, *query* and *value* encoders, each taking the form of a linear layer:

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

Where $\mathbf{x}$ is the output of the former encoder layer (for the first encoder layer $\mathbf{x}$ is the output of the embedding layer). These are then combined using an attention mechanism that does not involve new parameters:

$$\mathbf{h}_1^{\ell} = att\left(\mathbf{Q}^{1,\ell}, \mathbf{K}^{1,\ell}, \mathbf{V}^{1,\ell}, .., \mathbf{Q}^{m,\ell}, \mathbf{K}^{m,\ell}, \mathbf{V}^{m,l}\right)$$

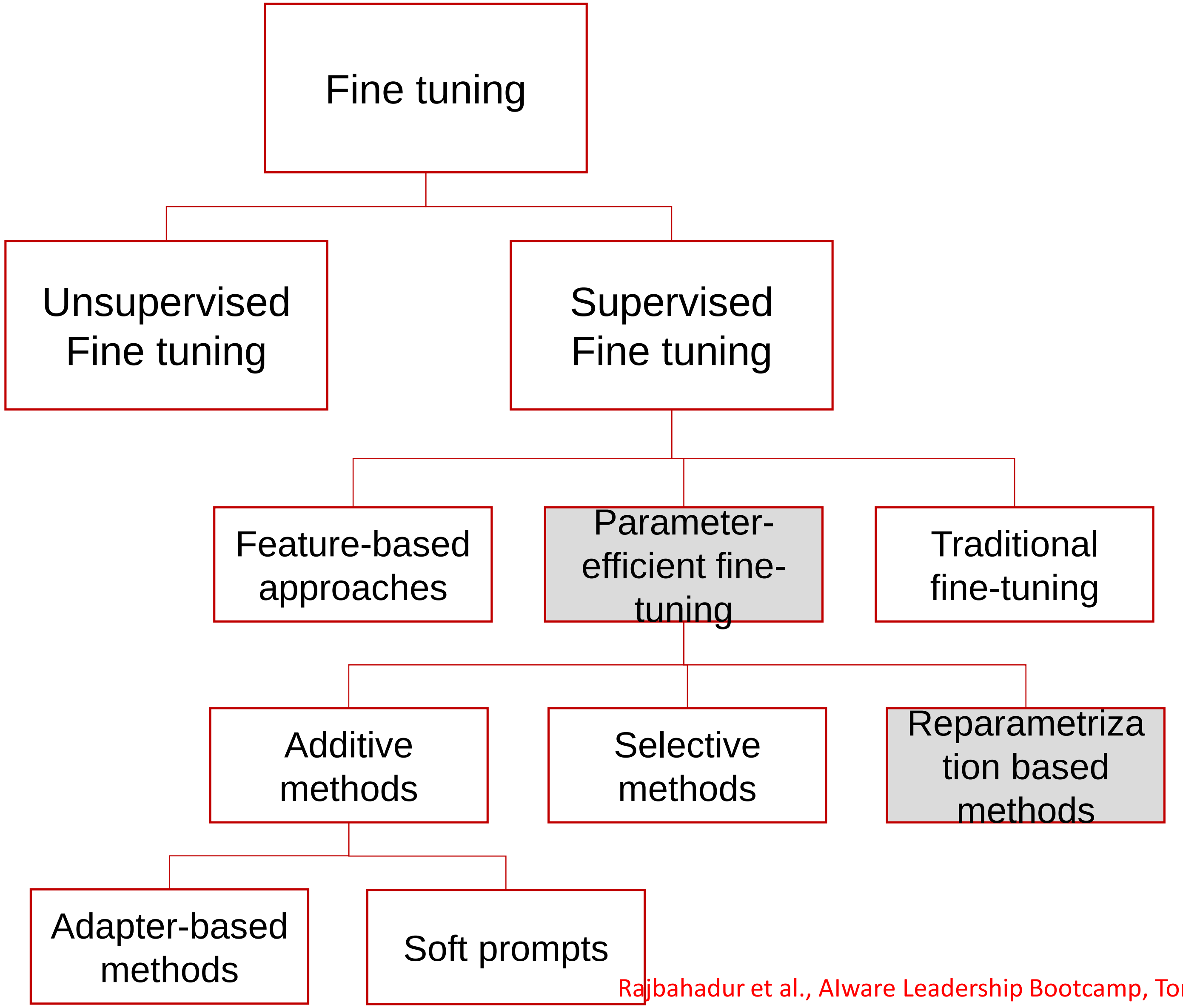
and then fed to an MLP with layer-norm (LN):

$$\mathbf{h}_2^{\ell} = \text{Dropout}\left(\mathbf{W}_{m_1}^{\ell} \cdot \mathbf{h}_1^{\ell} + \mathbf{b}_{m_1}^{\ell}\right) \quad (1)$$
$$\mathbf{h}_3^{\ell} = \mathbf{g}_{LN_1}^{\ell} \odot \frac{(\mathbf{h}_2^{\ell} + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^{\ell} \quad (2)$$
$$\mathbf{h}_4^{\ell} = \text{GELU}\left(\mathbf{W}_{m_2}^{\ell} \cdot \mathbf{h}_3^{\ell} + \mathbf{b}_{m_2}^{\ell}\right) \quad (3)$$
$$\mathbf{h}_5^{\ell} = \text{Dropout}\left(\mathbf{W}_{m_3}^{\ell} \cdot \mathbf{h}_4^{\ell} + \mathbf{b}_{m_3}^{\ell}\right) \quad (4)$$
$$\text{out}^{\ell} = \mathbf{g}_{LN_2}^{\ell} \odot \frac{(\mathbf{h}_5^{\ell} + \mathbf{h}_3^{\ell}) - \mu}{\sigma} + \mathbf{b}_{LN_2}^{\ell} \quad (5)$$

- BitFit is a selective approach that only tunes a selected subset of parameters from the FM.
- BitFit approach, freezes all the weights of the model and only tunes the bias terms to achieve a competitive performance.
- The Bias terms typically contribute upto 0.04% of all the model parameters which greatly cuts down the required resources.

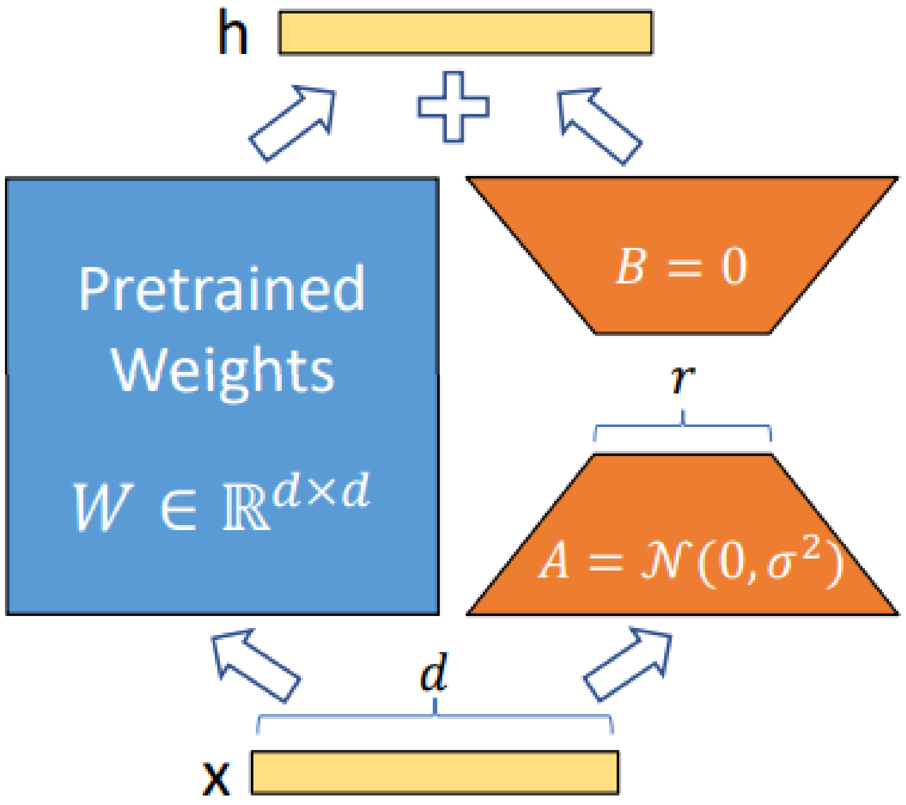
Zaken et al., BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-model

Taxonomy of Fine-tuning approaches



Parameter Efficient Fine-tuning approaches

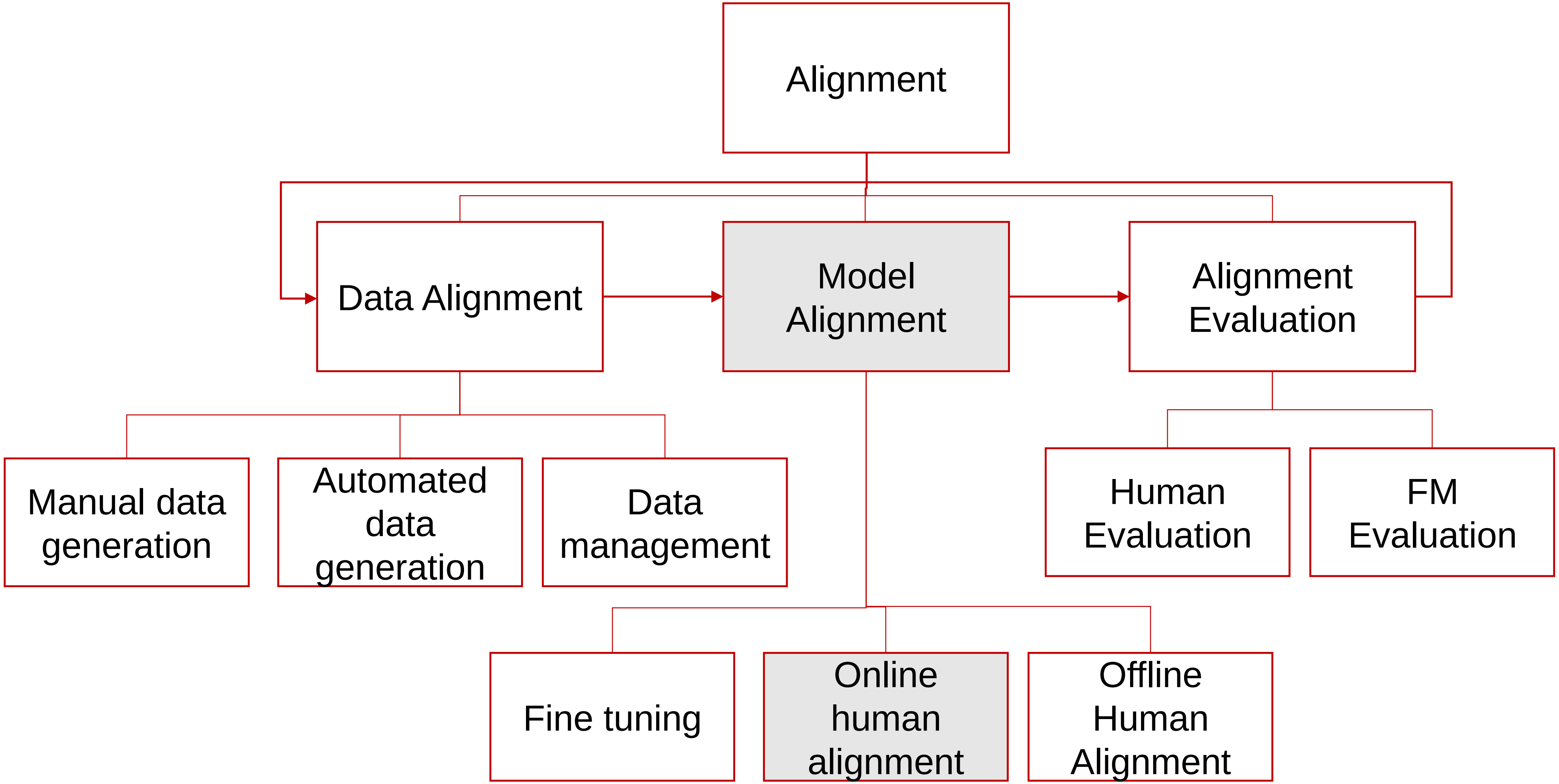
LoRA (Low Rank Adapatation)



- LoRA introduces low-rank matrices to the model's layers, allowing only these smaller matrices to be trained while keeping the original model weights frozen.
- For each weight matrix **W**, LoRA decomposes it into two low rank matrices **A** and **B.**
- During fine-tuning only **A** and **B** are updated
- Reduces the memory consumption
- LoRA is currently one of the most widely adopted supervised fine tuning methods since they reduce the hardware requirement by upto 3 times and there is no inference latency

Hu et al., LoRA: Low-Rank Adaptation of Large Language Models

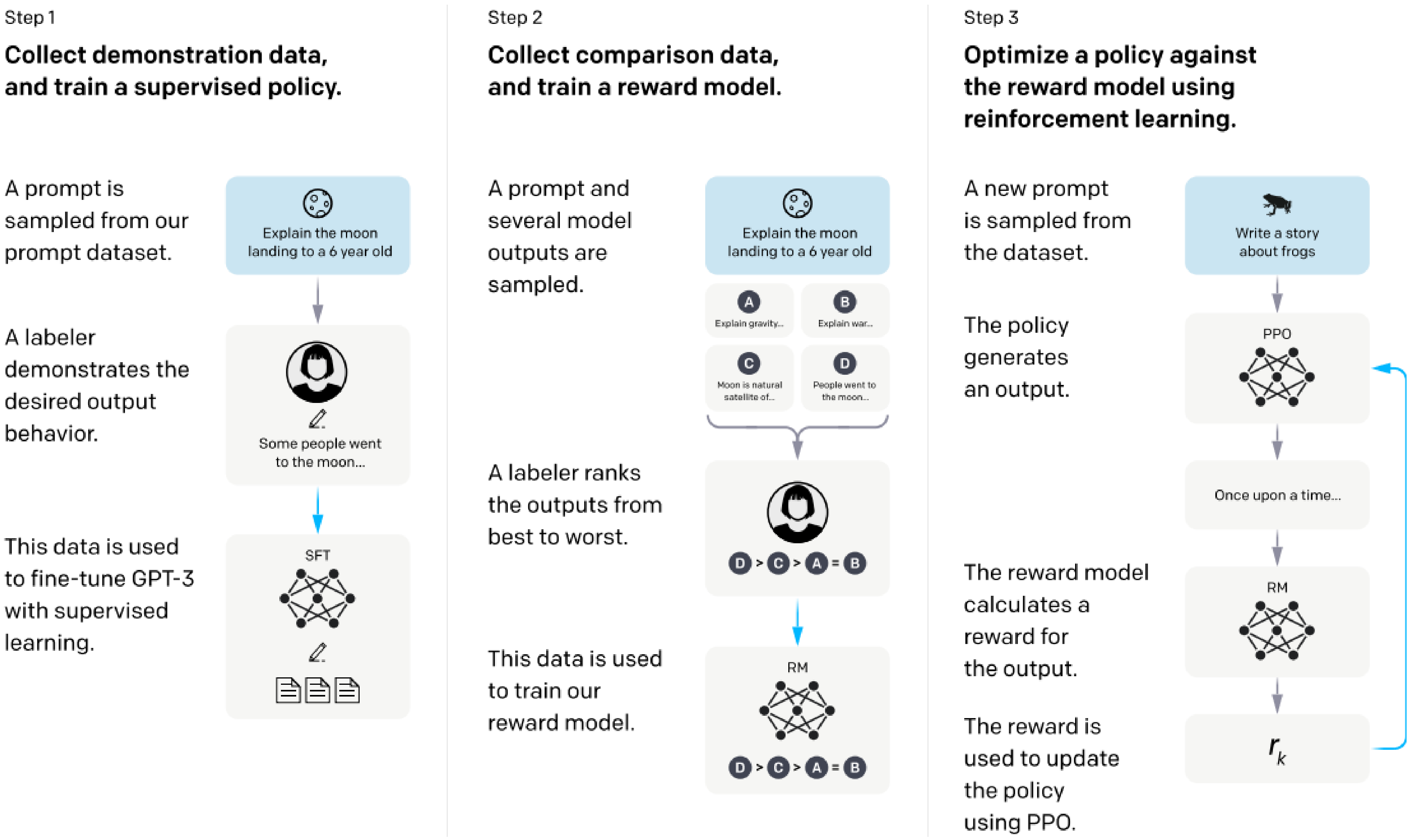
Taxonomy of Alignment Engineering



Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

Online Human Alignment

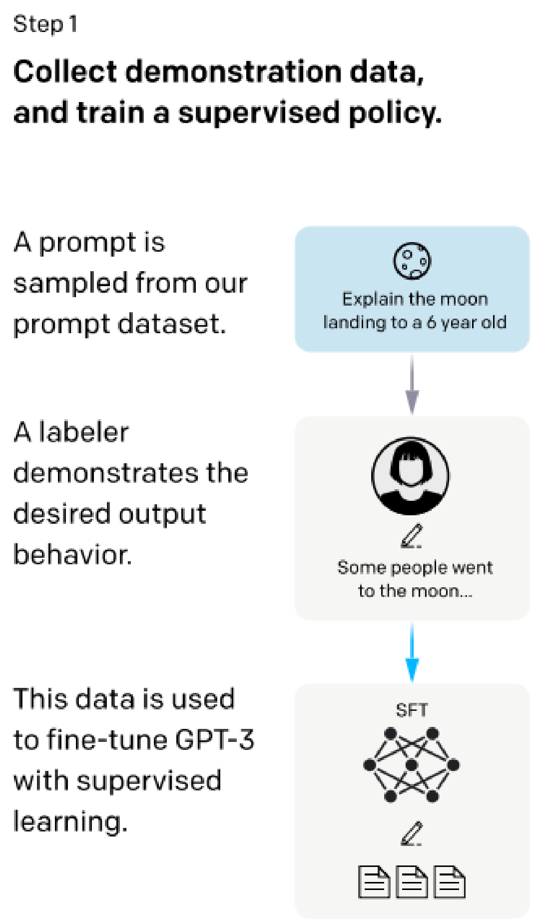
Reinforcement Learning with Human Feedback (RLHF)



Ouyang et al., Training language models to follow instructions with human feedback

Rajbahadur et al., AIware Leadership Bootcamp, Toronto, Canada, 2024

Reinforcement Learning with Human Feedback (RLHF)



Purpose: The model undergoes initial alignment with human-like behavior through supervised training on specific examples.

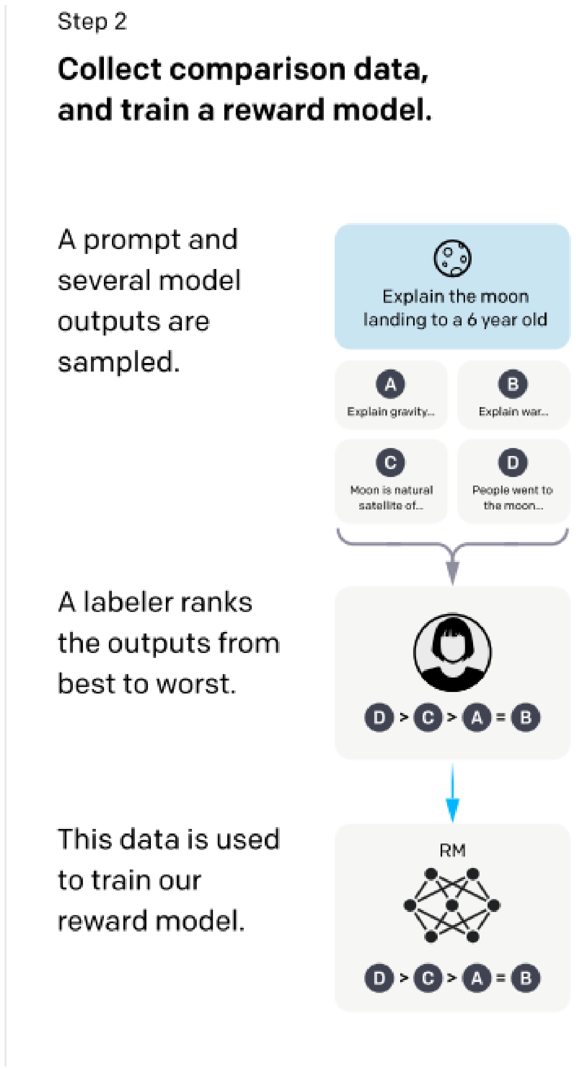
Process: Human annotators create or curate examples of high-quality responses for various prompts.

- These responses are used to fine-tune the pretrained model in a supervised manner. This stage creates a foundation for better responses and primes the model for further alignment.

Outcome: The model learns basic patterns of preferred responses, which makes it more likely to generate coherent, relevant answers even before reinforcement learning is applied.

Online Human Alignment

Reinforcement Learning with Human Feedback (RLHF)



Purpose: To provide a mechanism for scoring responses based on human preferences.

Process:

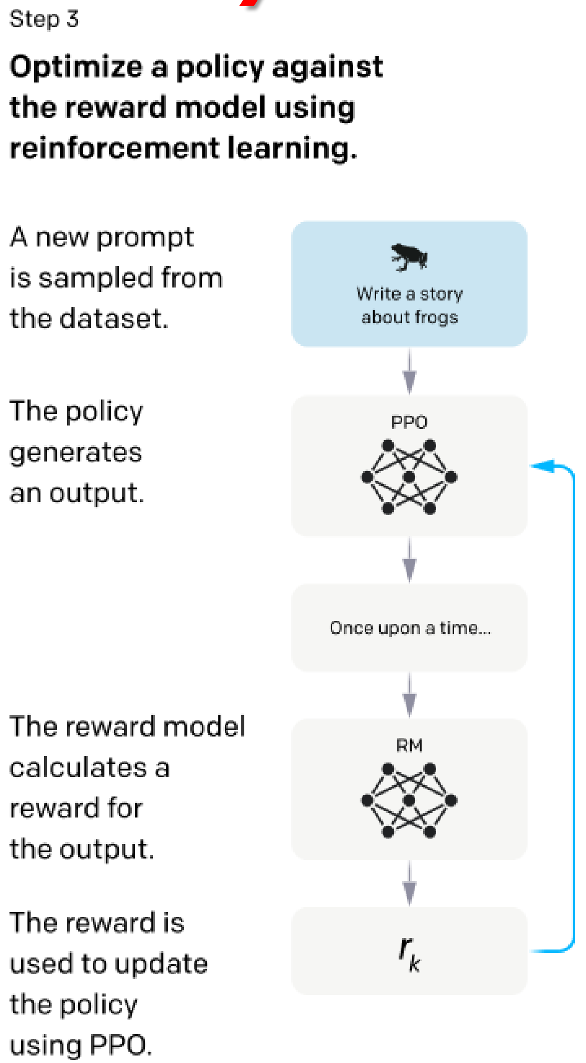
Human Feedback Collection: Human evaluators rank several model-generated responses for each prompt, from best to worst, based on qualities like clarity, helpfulness, and alignment with human expectations.

Training the Reward Model: Using these rankings, a reward model is trained to predict a reward score that represents how well a response aligns with human preferences. Using these rankings, a reward model is trained to predict a reward score that represents how well a response aligns with human preferences.

Outcome: The reward model generalizes human preferences, enabling it to evaluate new responses by assigning reward scores without direct human input every time.

Online Human Alignment

Reinforcement Learning with Human Feedback (RLHF)



Purpose: To refine the model's behavior by maximizing alignment with human preferences, as captured by the reward model.

Process:

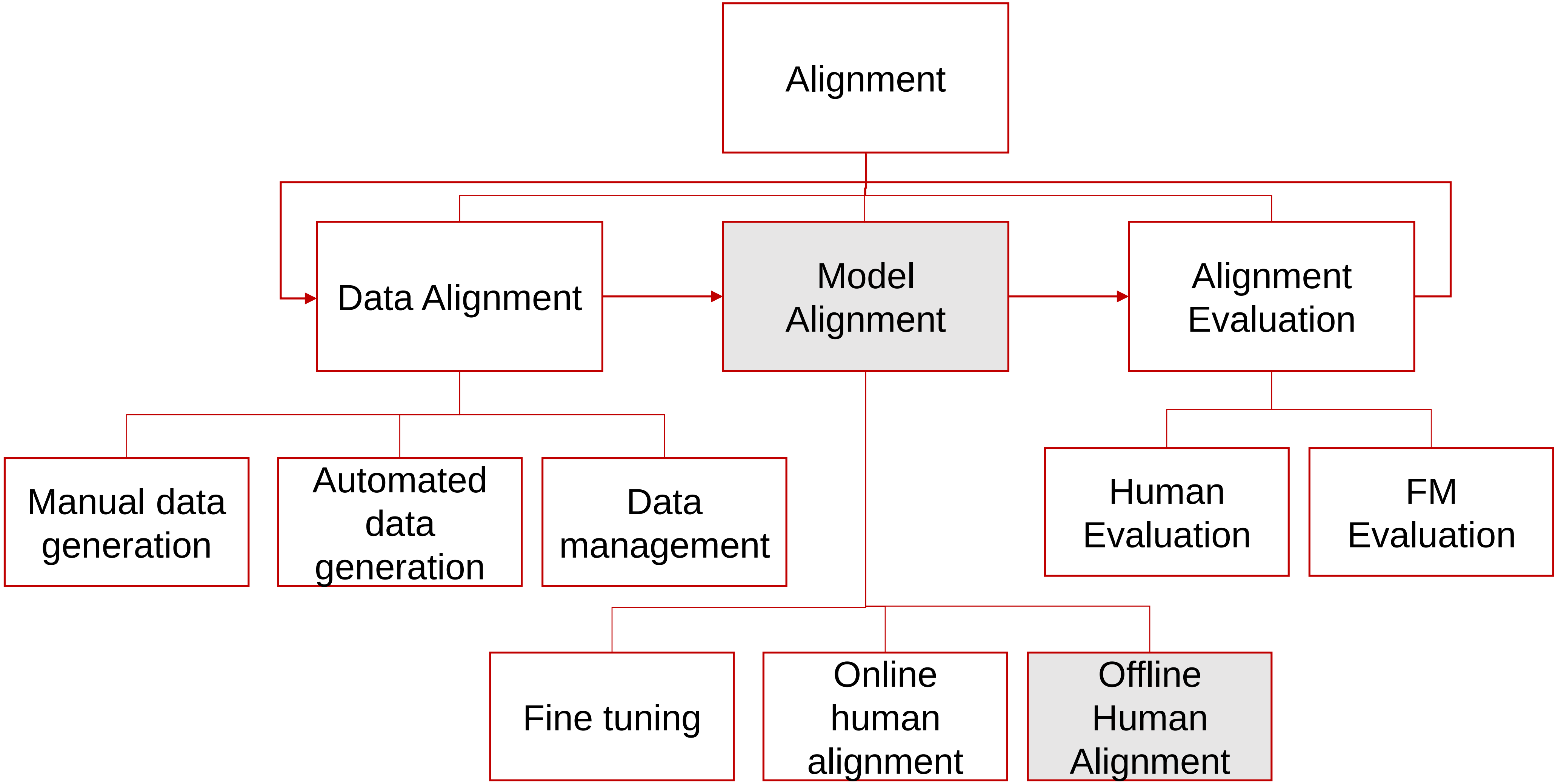
Generating Responses: The model generates a response to a prompt, simulating real-world interaction.

Reward Calculation: The reward model assigns a score to the generated response, reflecting how well it aligns with human preferences learned in the previous step.

Policy Optimization (PPO): Proximal Policy Optimization (PPO) is applied to adjust the model's parameters based on the reward score. The model uses policy loss to boost high-reward responses, value loss to stabilize predictions, and entropy loss to encourage diversity, iteratively refining itself through feedback from the reward model.

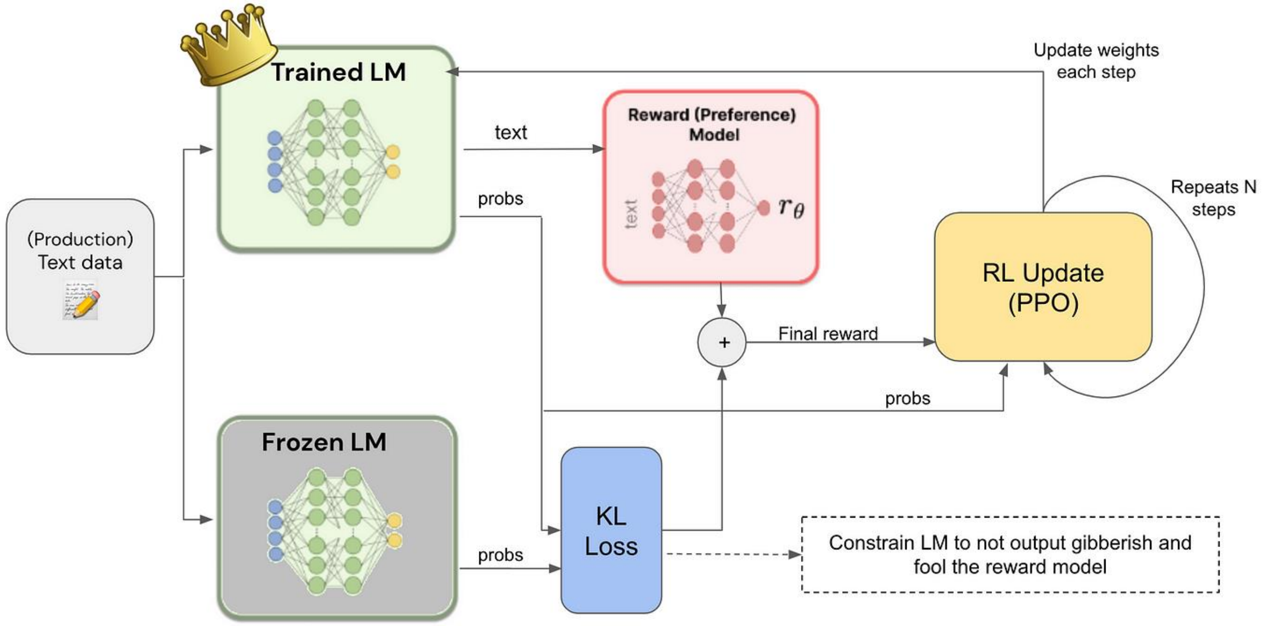
Outcome: The model becomes adept at producing responses that are relevant, clear, and aligned with human expectations. Through this iterative feedback loop, the model learns to prioritize responses that maximize human-aligned rewards

Taxonomy of Alignment Engineering



Offline Human Alignment

Direct Preference Optimization



Purpose: To align model outputs with human preferences by using a reward model directly

Process:

Collect Human Preferences: Human evaluators rank multiple responses for each prompt.

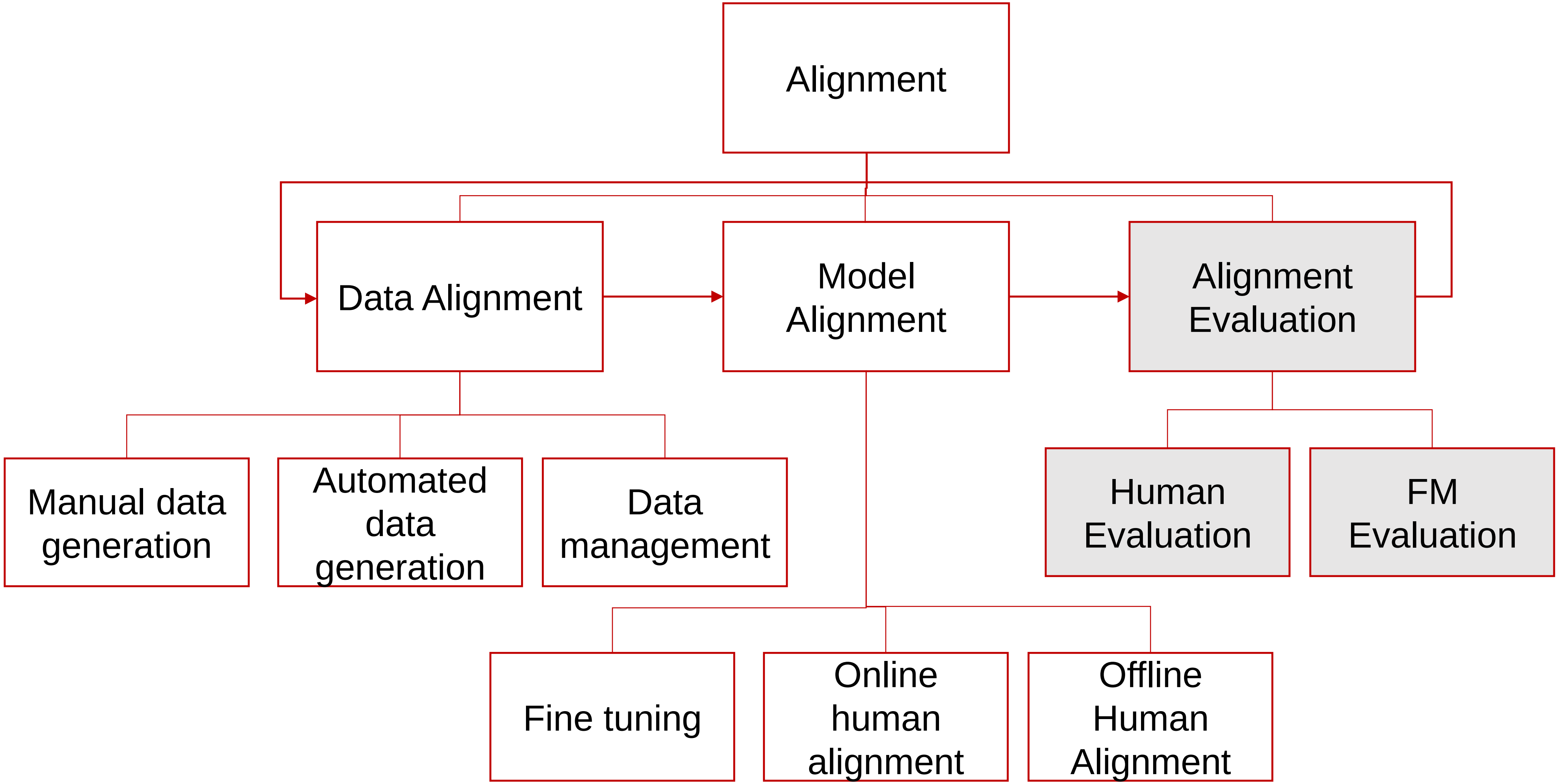
Train a Reward Model: A reward model is trained to predict preference scores based on the human rankings, similar to RLHF.

Direct Optimization Using Preference Scores: Instead of using reinforcement learning, DPO directly fine-tunes the main model by adjusting its outputs based on the reward model's preference scores.

Outcome: The model produces responses that align closely with human preferences while making alignment more computationally efficient.

Rafailov et al., Direct Preference Optimization: Your Language Model is Secretly a Reward Model

Taxonomy of Alignment Engineering



Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

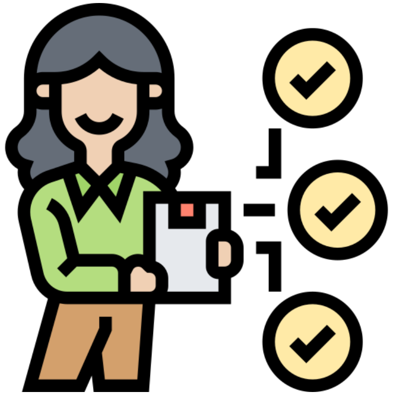
Alignment Evaluation

Benchmarks

Use existing benchmarks

Benchmarks are not very representative of real-world tasks and lots of data leakage

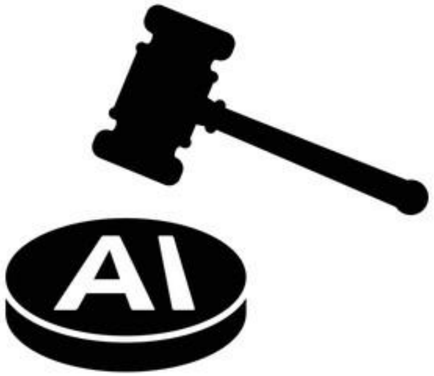
Human Judges



Using human judges to evaluate the quality of alignment; Use ELO ratings to rank the judges

Can be very subjective and subject to human bias and not very scalable

AI Judges

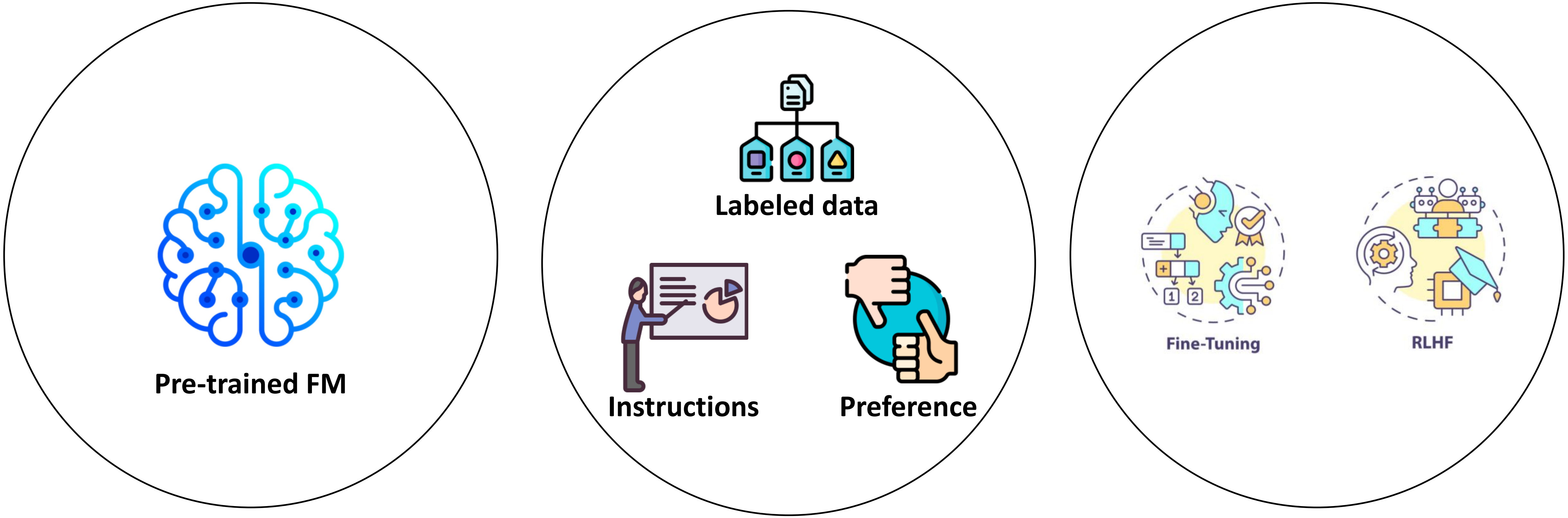


Using FMs to evaluate the alignment quality

Can be very costly and FMs can be very unreliable judges

Remember this slide?

Ingredients required for Alignment Engineering



Data to
teach the FM

Method to
teach the FM

Remember this slide?

Ingredients required for Alignment Engineering

Is throwing a bunch of data at the FM the best way to teach the FM to excel?

We don't just throw a bunch of random concepts at students when we are teaching them a course! We use a **curriculum** to structure and enable the learning!

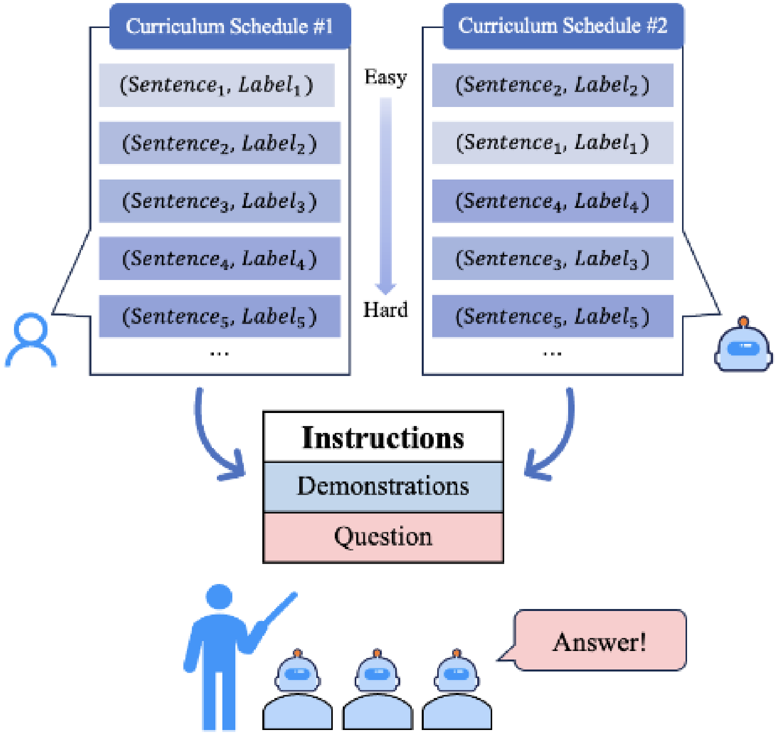
Data to teach the FM

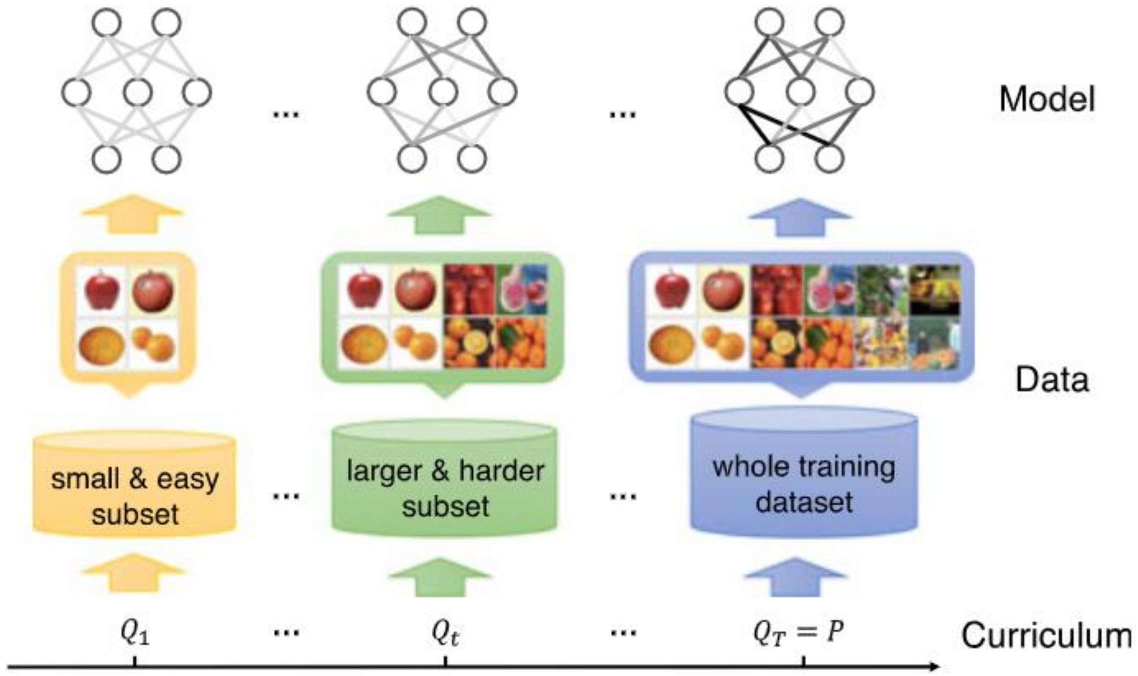
Method to teach the FM

Overview of the session

- ❑ **A brief intro to pre-training Foundation Models (FM):** An introduction to how a FM is pre-trained
- ❑ **Why do we have to align FMs:** Motivating the need for Alignment
- ❑ **Taxonomy of Alignment Engineering**
  - ❑ Data Alignment
  - ❑ Model Alignment
    - ❑ Finetuning
    - ❑ Online alignment
    - ❑ Offline alignment
  - ❑ Alignment Evaluation
- ❑ **Curriculum Learning**

What is curriculum learning?



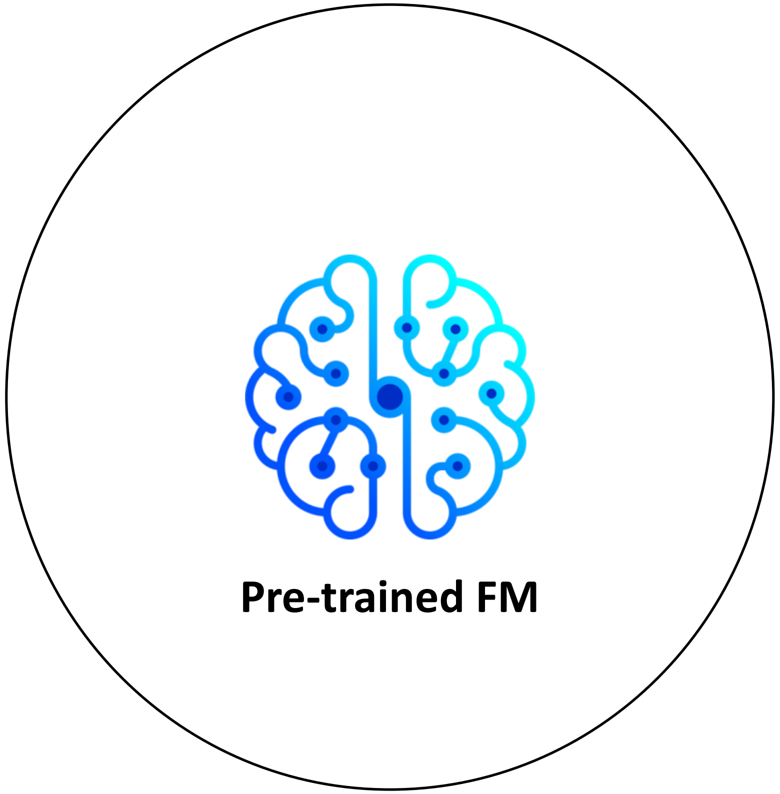


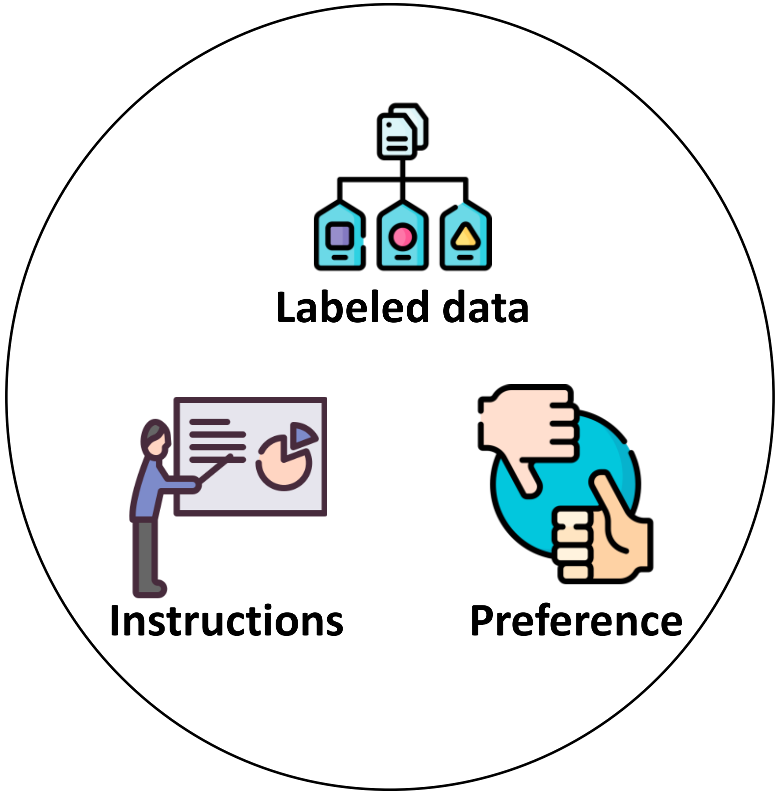
Curriculum learning is a training approach where the model is introduced to data in a **progressively complex order**, starting from simpler tasks and moving to more challenging ones.

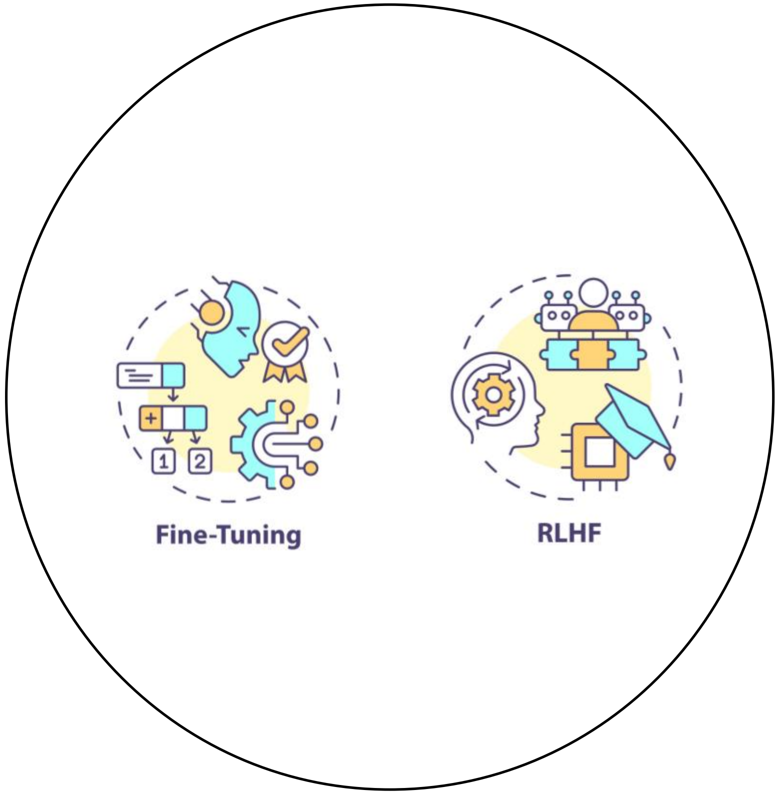
This helps the model build a solid understanding of basic patterns before tackling harder language structures, improving learning efficiency and robustness.

Liu et al., Let's Learn Step by Step: Enhancing In-Context Learning Ability withCurriculum Learning; Wang et al., A Survey on Curriculum Learning

Leveraging Curriculum Learning for better Alignment Engineering







Systematically curate the data to form a curriculum composed of the knowledge that we want the FM to learn

Incrementally expose the model to data based on our curriculum

Leveraging Curriculum Learning for better Alignment Engineering

Pre-trained FM

Curriculum is a new asset like code and as Software Engineering practitioners and researchers, we should focus on this part to build better FMware to support SE3.0!

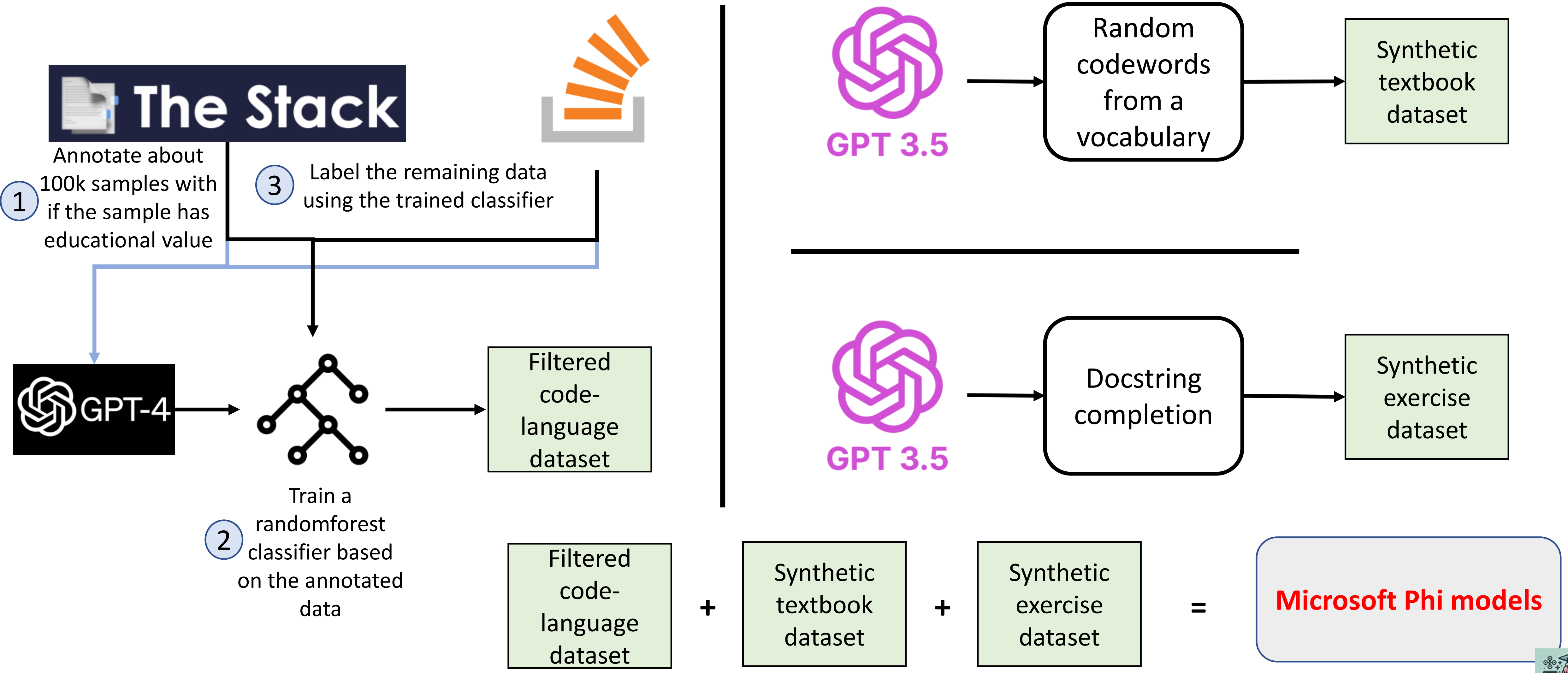
Lets leave this part to the AI folks now ;-)

Systematically curate the data to form a curriculum composed of the knowledge that we want the FM to learn

Incrementally expose the model to data based on our curriculum

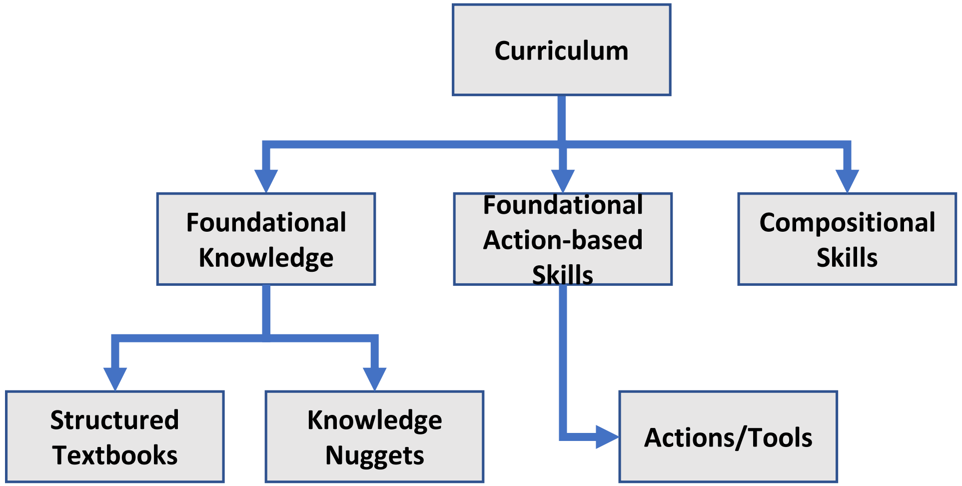
Curriculum learning

Microsoft Phi Models



Gunasekar et al., Textbooks Are All You Need

While Phi approach is great, it needs to be more systematic



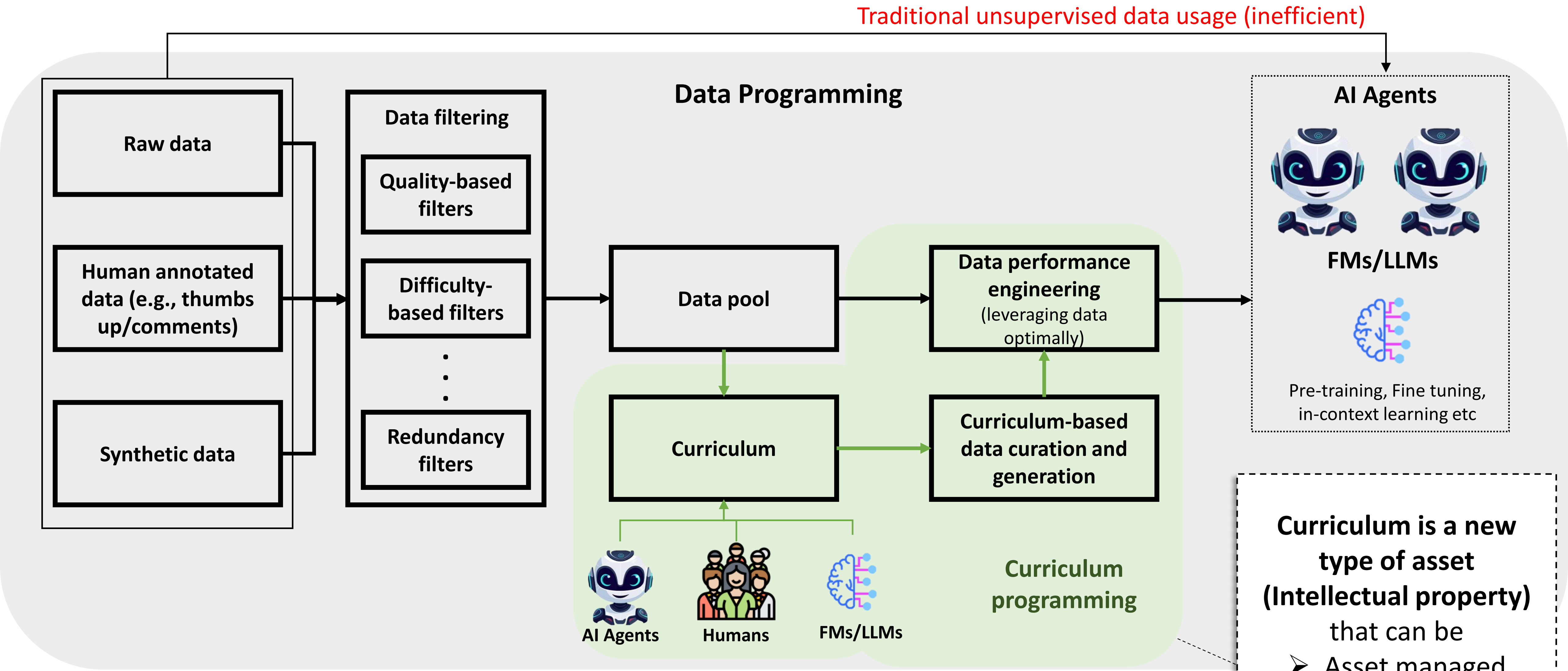
Key goals

1) Curriculum creation should be collaborative to maximize reuse and efficiency across an organization
2) Curriculum creation can be AI assisted (e.g., Phi-coding text books are auto-generated) but best to have AI+Human collaborate with Human as reviewer and architect
3) Too low level skills lead to agents that take too long to discover new compositional skills (as noted by Voyageur and SWEAgent)
4) Compositional skills are ideally learned otherwise Agents will stagnate

Need SE Technology for Curriculum Engineering

1) Curriculum co-creation and versioning technology
2) Curriculum quality reviewing technology
3) Curriculum QA technology (e.g., optimization by removing redundant information, or repair by removing correct or outdated information)
4) Curriculum to synthetic data creation technology

Data programming and Curriculum programming – a proposal



Curriculum is a new type of asset (Intellectual property) that can be

- Asset managed
- Reused
- ...

Using Data as-is is a very inefficient, costly, and ad-hoc mean to teach our SE AI Teammates (AIware). Instead, data programming and curriculum programming are more effective and efficient ways of teaching our AI SE Teammates

Curriculum programming for code documentation

Code documentation curriculum

Creates

AI Agents Humans FMs/LLMs

By adding

Foundational Knowledge

- Documentation standards
- Common pitfalls when creating documentation
- Code design principles

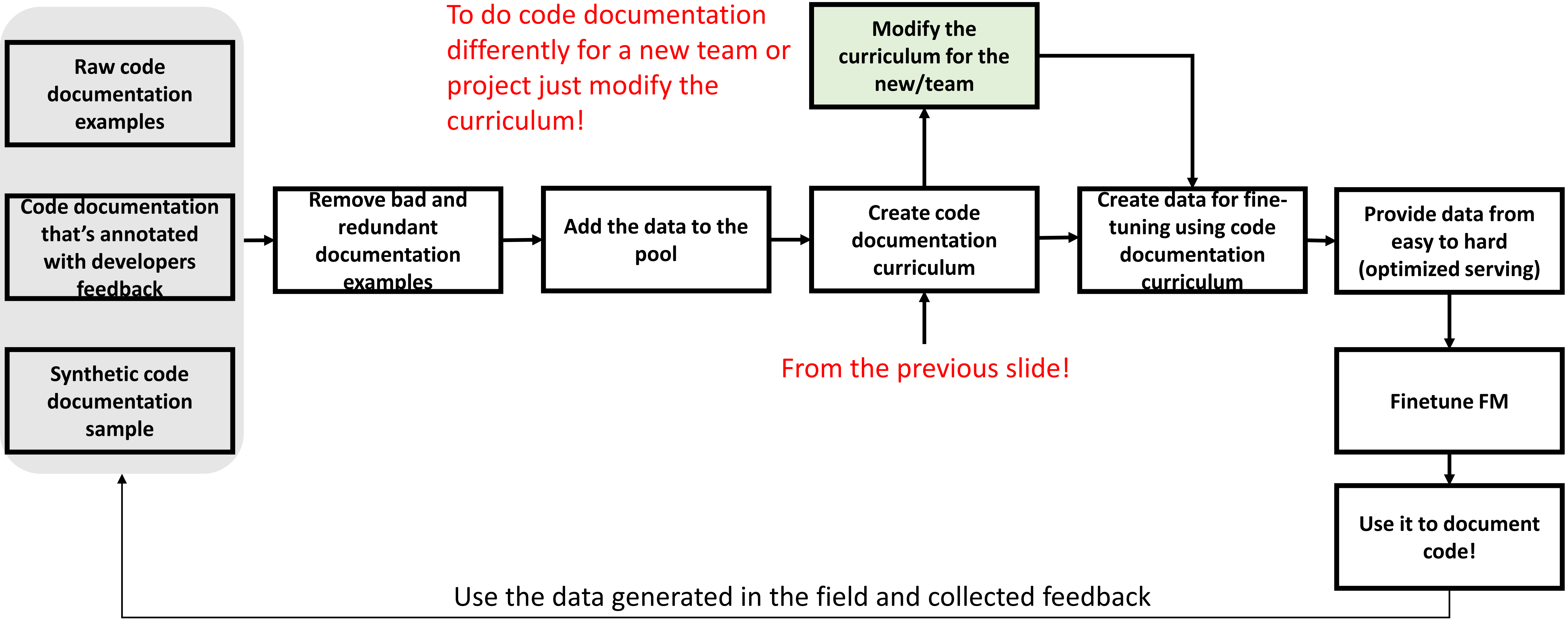
Foundational Action-based Skills

- Technical writing (e.g., detailing the purpose of the parameters of a function)
- Code anlaysis
- Extracting information from the version control system

Compositional Skills

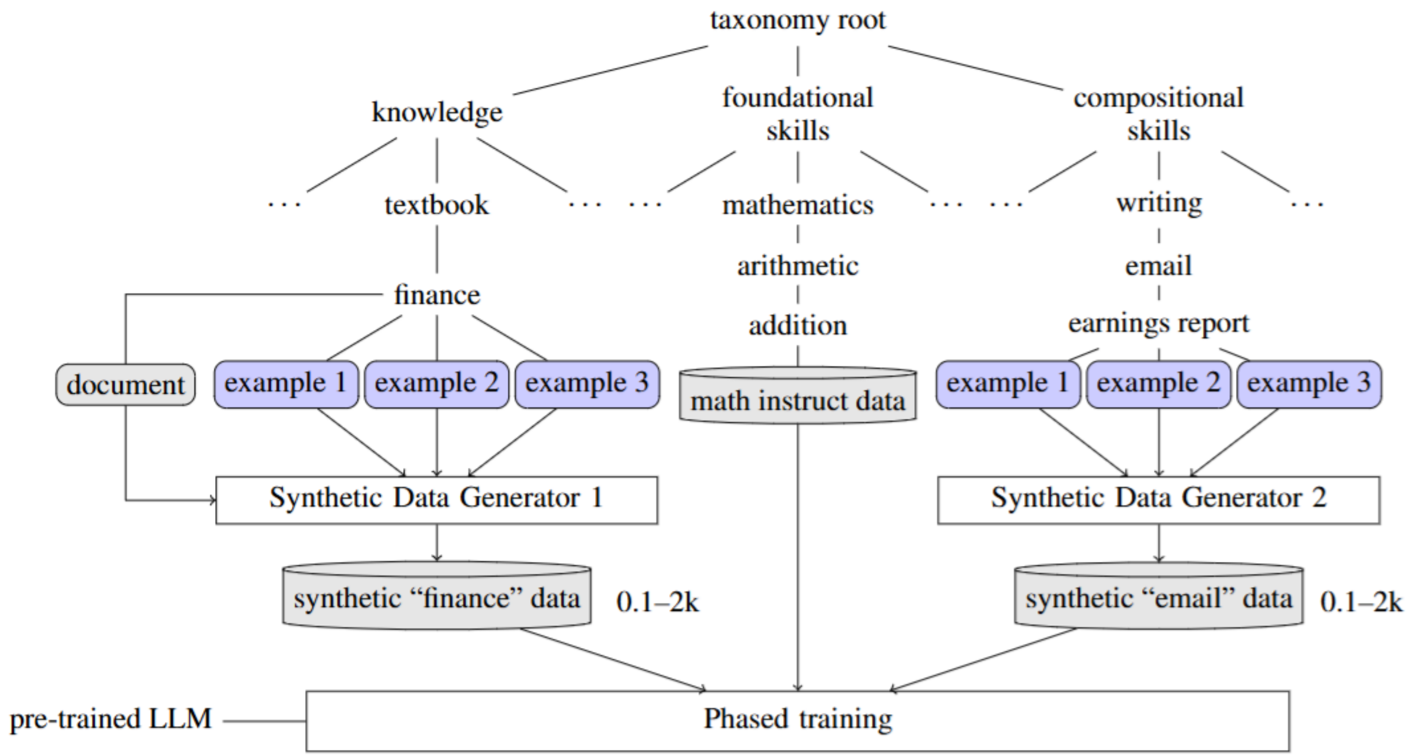
- Integrating code and documentation
- Adapting the documentation for different audiences
- Ability to combine code and documentation from different files

Data programming for code documentation



Curriculum learning

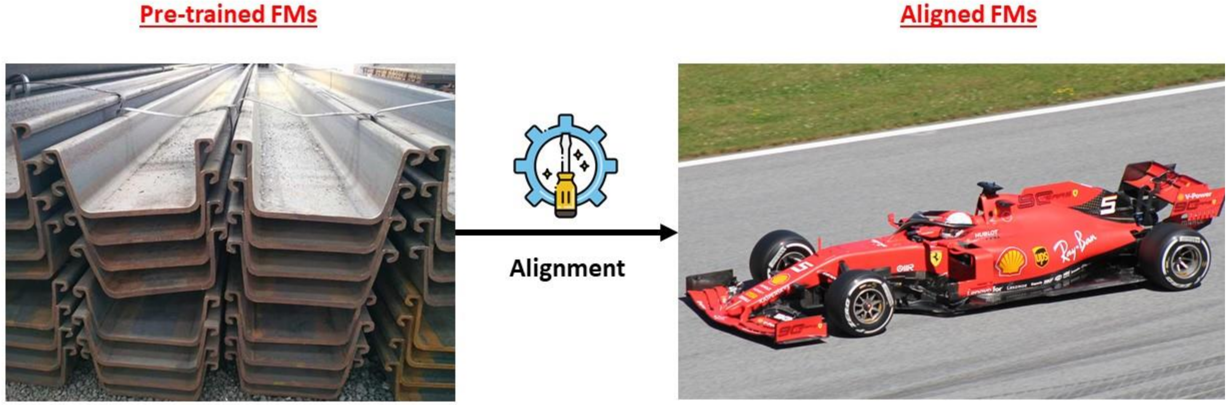
IBM InstructLab



- Decompose knowledge into Knowledge, foundational skills and compositional skills
- Leverage crowdsourcing to collect such Knowledge, skills and compositional skills for multiple domains in the form of question and answer pairs
- These skills and knowledge together acts as the curriculum for synthetic data generation
- The synthetically trained data is then used to fine-tune the model in a two-phase process

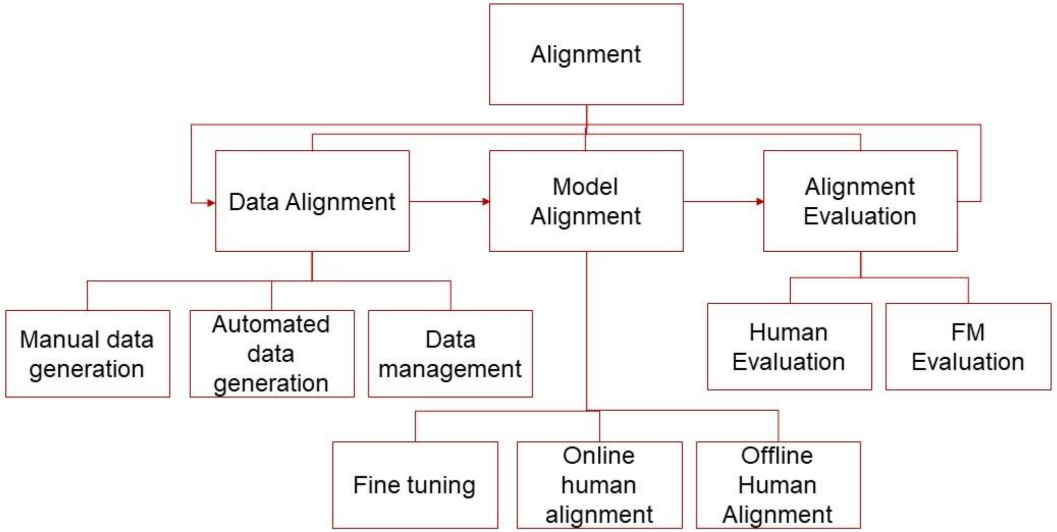
Xu et al., LAB: Large-Scale Alignment for ChatBots

Pretrained (FMs) Isn't Practical: Alignment for Real-World Use



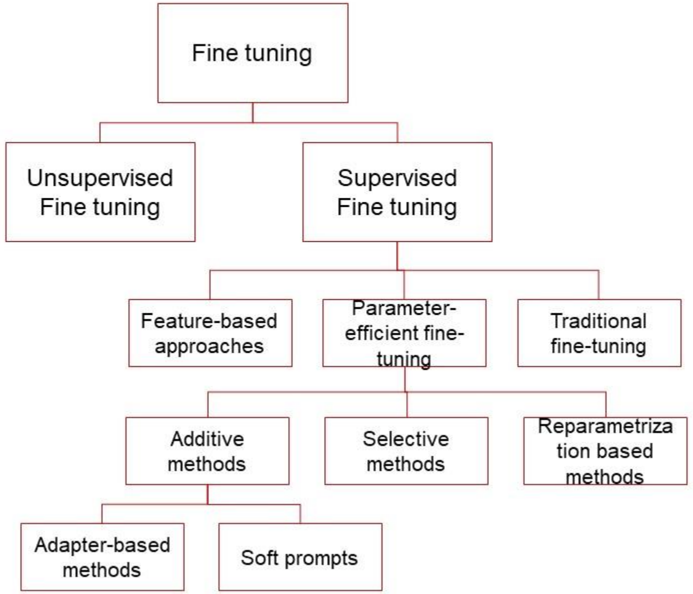
The process of adjusting and guiding a FM's behavior through fine-tuning, prompt engineering, reinforcement learning from human feedback (RLHF), and other methods to ensure **it meets specific objectives, values, and safety standards for practical use.**

Taxonomy of Alignment Engineering

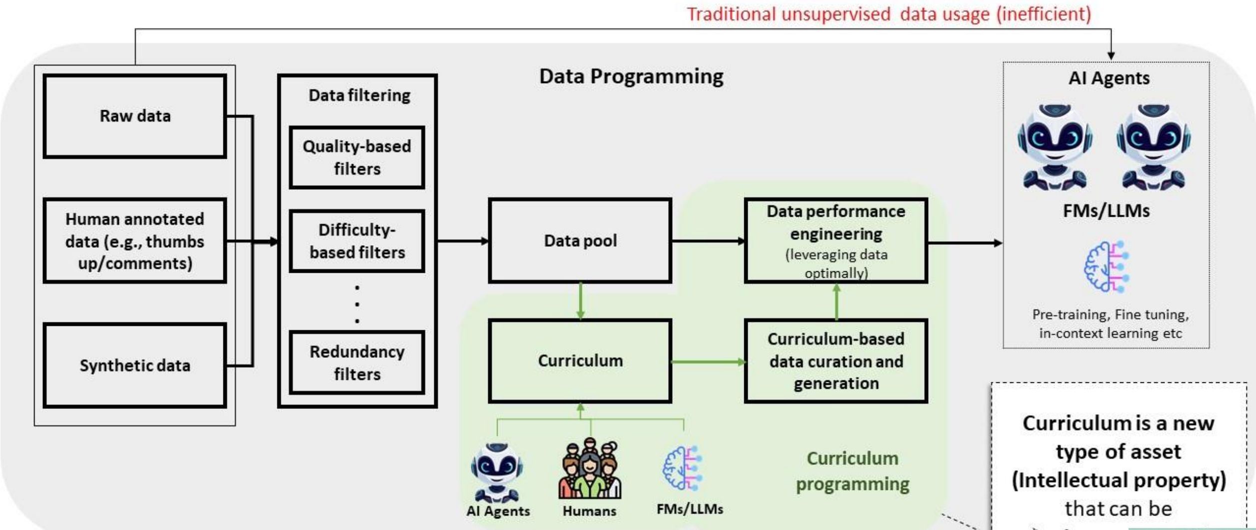


Inspired from Wang et al., Aligning Large Language Models with Human: A Survey

Taxonomy of Fine-tuning approaches



Data programming and Curriculum programming – a proposal



Curriculum is a new type of asset (Intellectual property) that can be
- ➢ Asset m
  - ➢ Reu
  - ➢

Using Data as-is is a very inefficient, costly, and ad-hoc mean to teach our SE AI Teammates (AIware). Instead, data programming and curriculum programming are more effective and efficient ways of teaching our AI SE Teammates

Pretrained (FMs) Isn't Practical: Alignment for Real-World Use

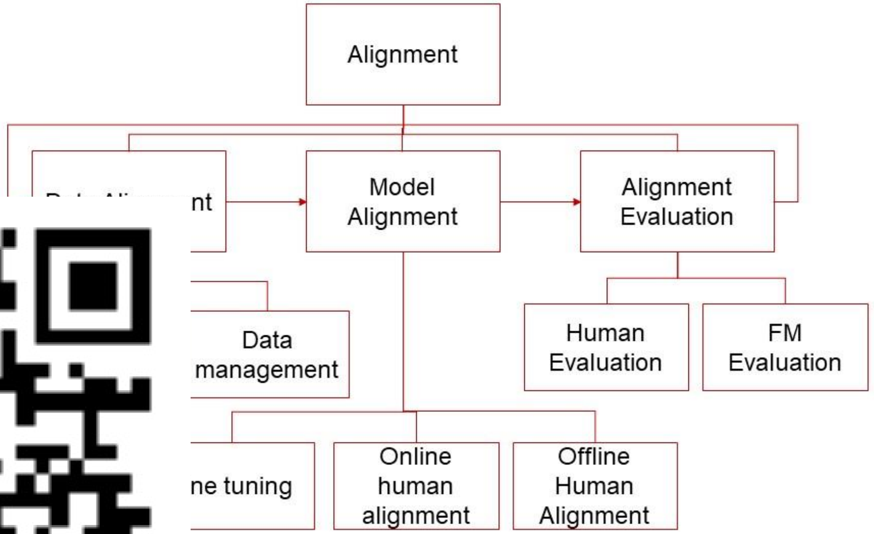
Pre-trained FMs

Aligned FMs

Alignment

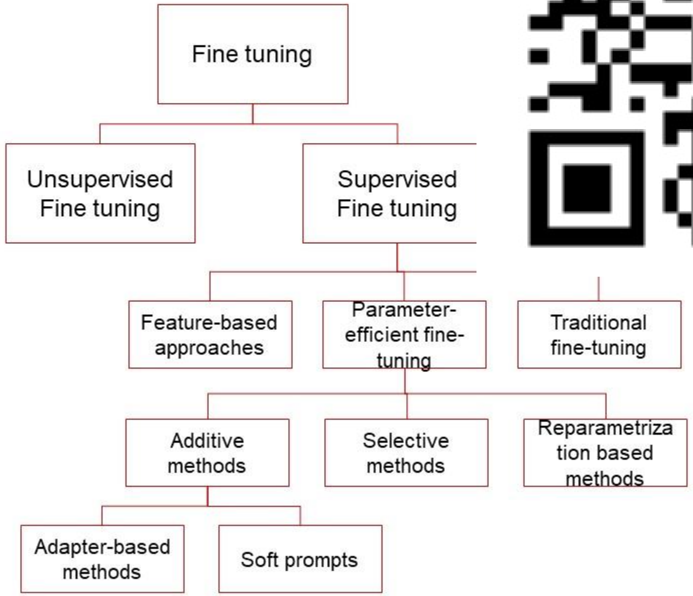
The process of adjusting and guiding a FM's behavior through fine-tu
reinforcement learning from human feedback (RLHF), and other metho
objectives, values, and safety standards for pract

Taxonomy of Alignment Engineering

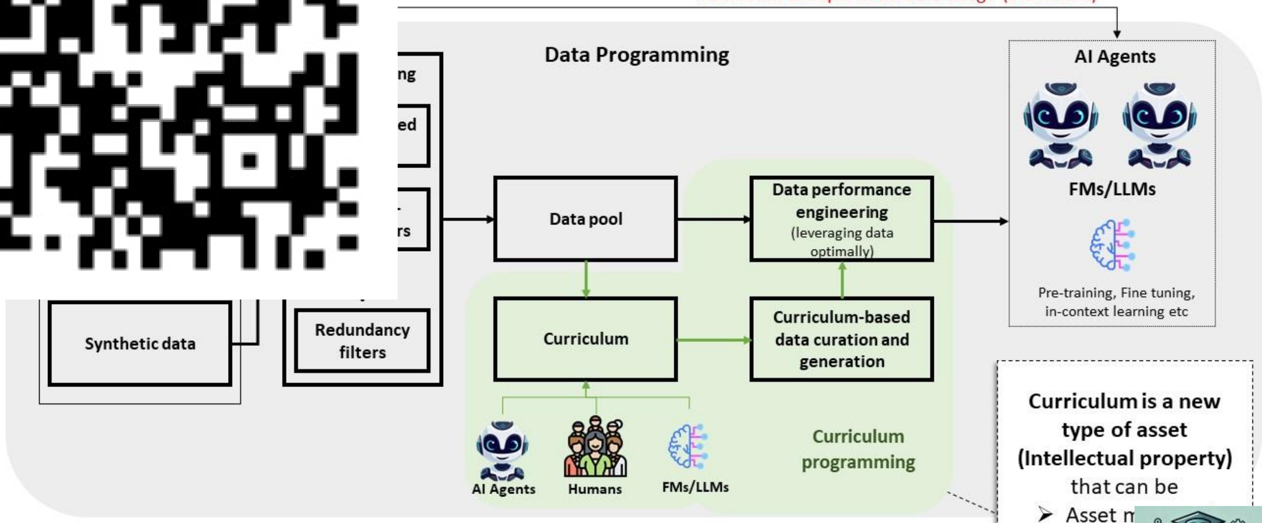


ɫls with Human: A Survey

Taxonomy of Fine-tuning appr



and Curriculum programming – a proposal



Curriculum is a new type of asset (Intellectual property) that can be
- Asset m
  - Reu

Using Data as-is is a very inefficient, costly, and ad-hoc mean to teach our SE AI Teammates (AIware). Instead, data programming and curriculum programming are more effective and efficient ways of teaching our AI SE Teammates