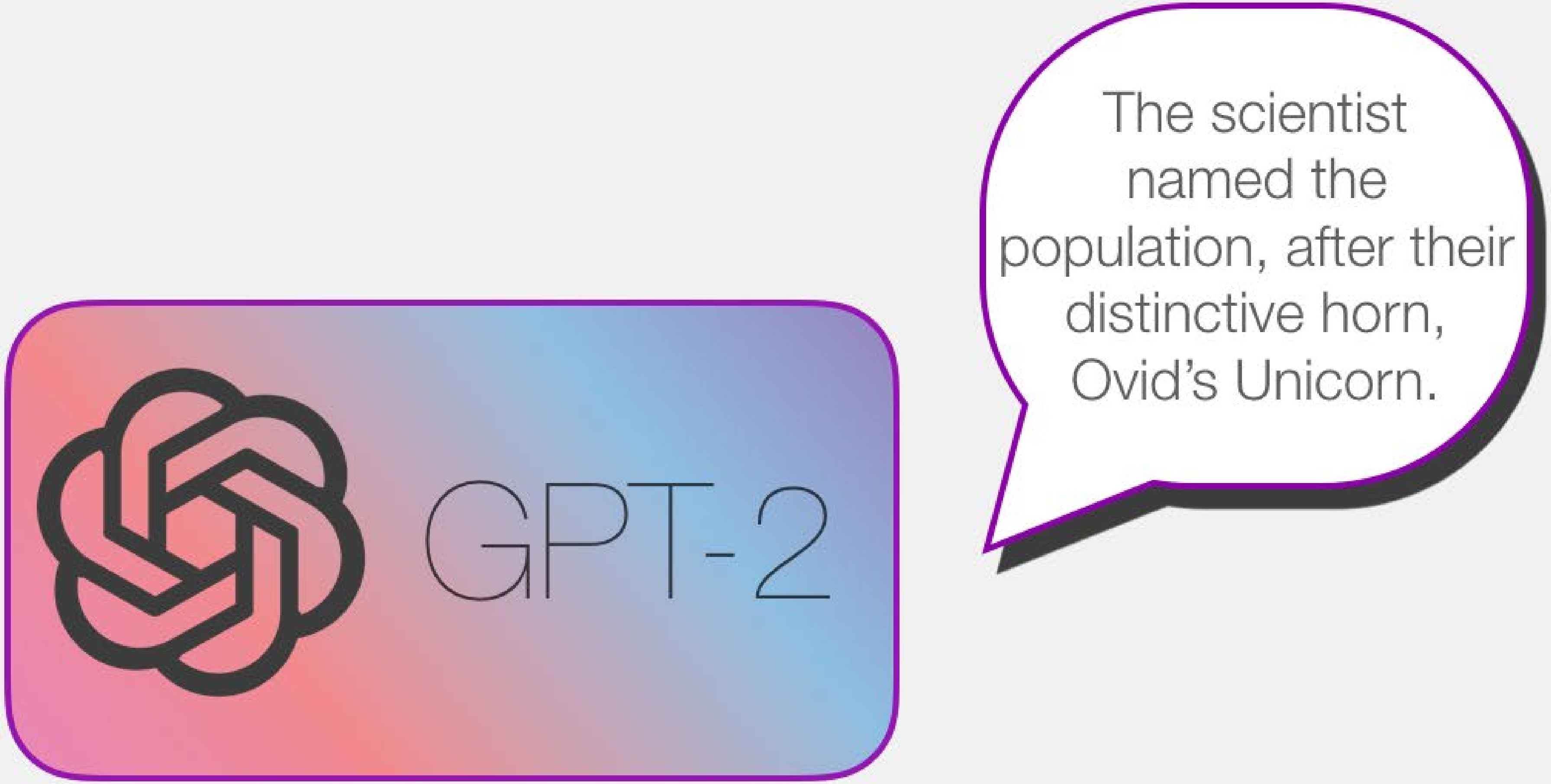


GPT2： Zero Shot的崛起

Language Model

input/feature #1 input/feature #2 output/label

Thou shalt ______

Input
Features

Output
Prediction

Thou → Trained Language Model

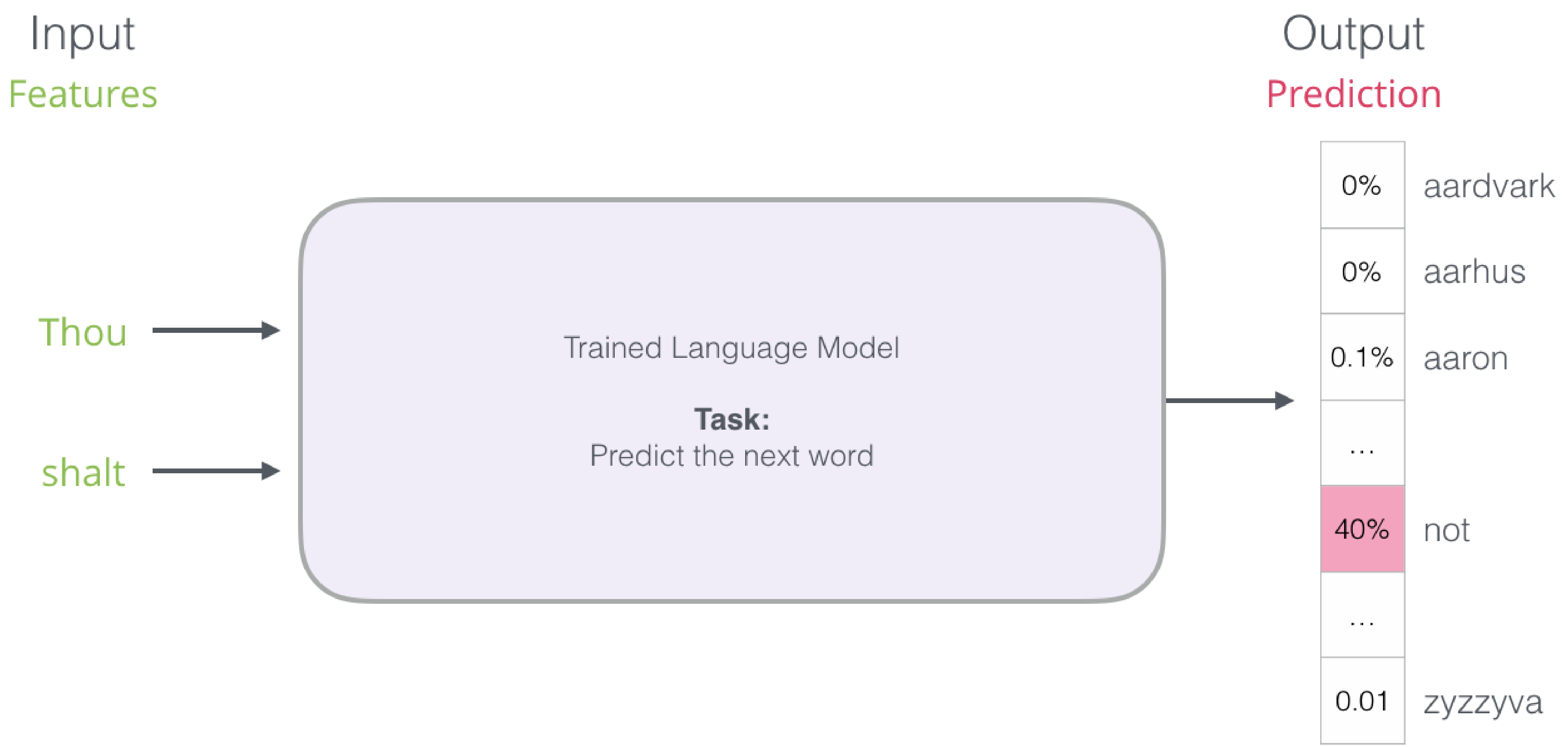
Task:
Predict the next word

shalt →

→ not

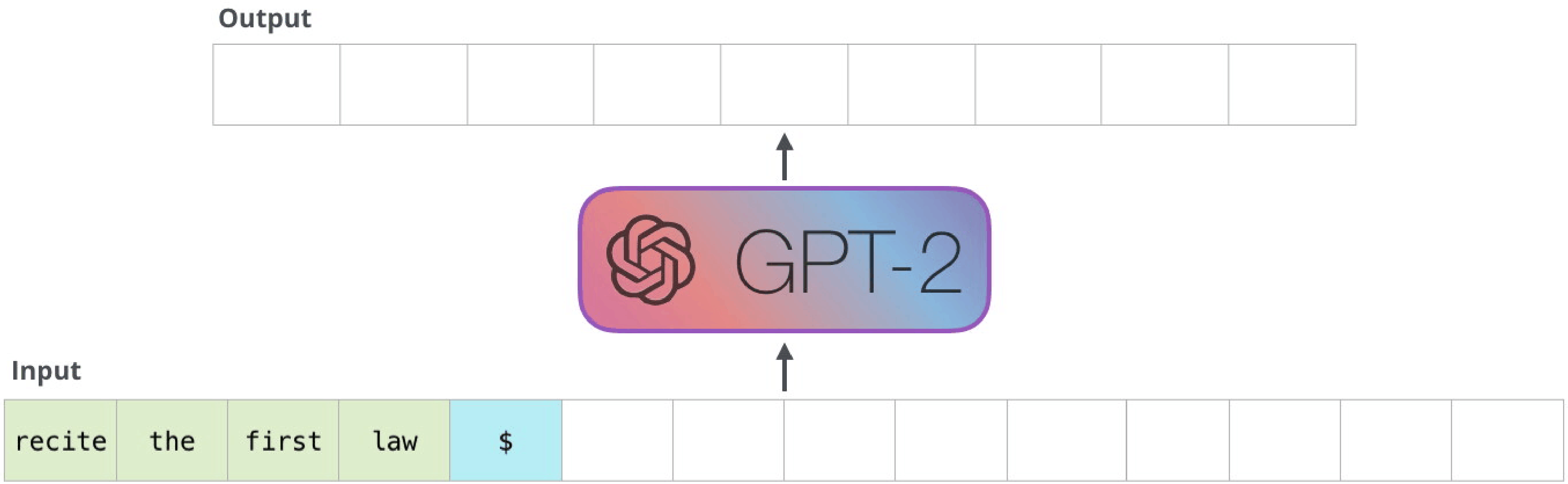
https://jalammar.github.io/illustrated-word2vec/

Language Model



https://jalammar.github.io/illustrated-word2vec/

Auto Regressive



Learning Objectives and Concepts

- **Task Conditioning**
- **Zero Shot Learning and Zero Short Task Transfer**

Task Conditioning

Training objective of language model is formulated as:

$$P(output|input)$$

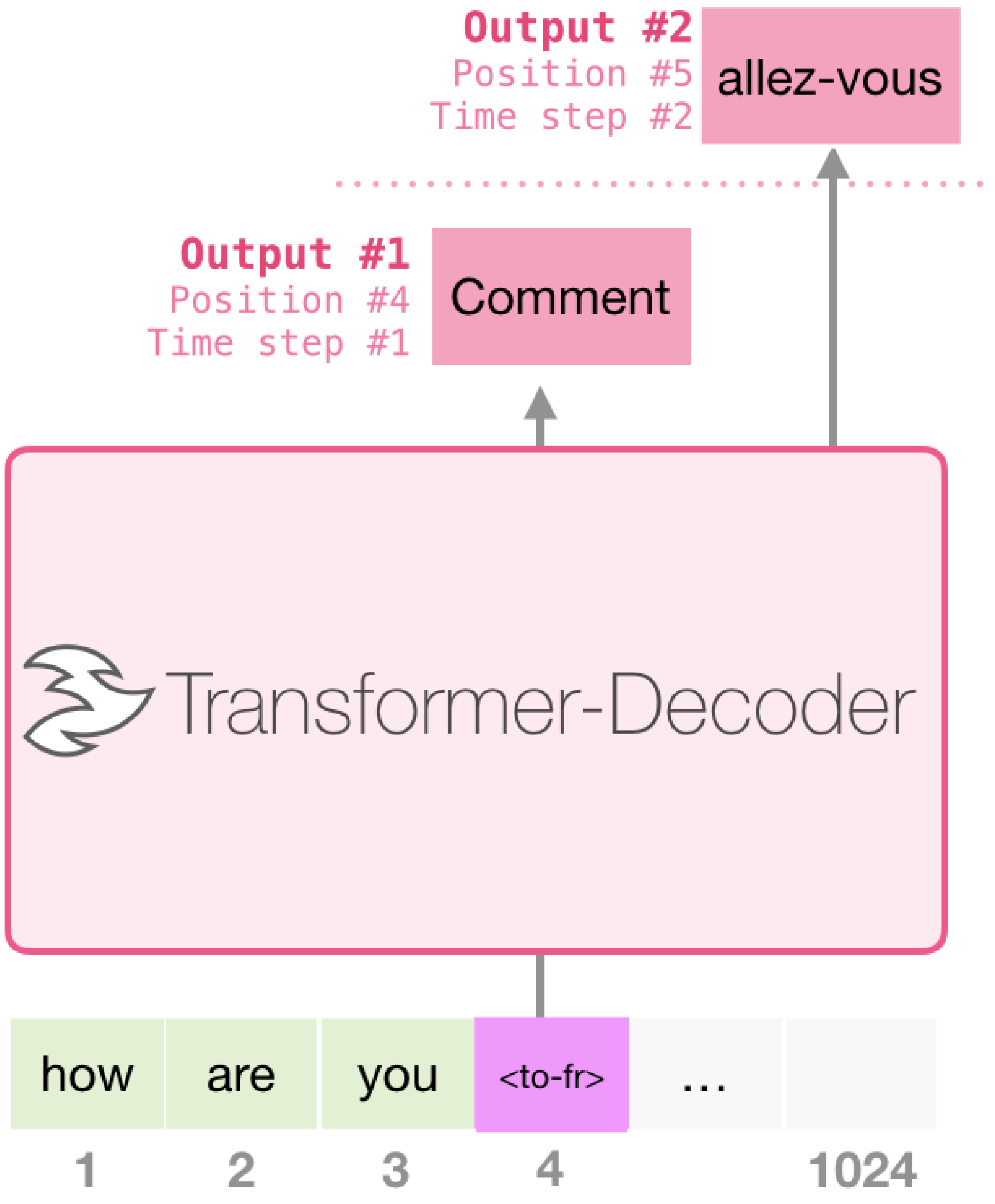
However, GPT-2 aimed at learning multiple tasks using the same unsupervised model. To achieve that, the learning objective should be modified to:

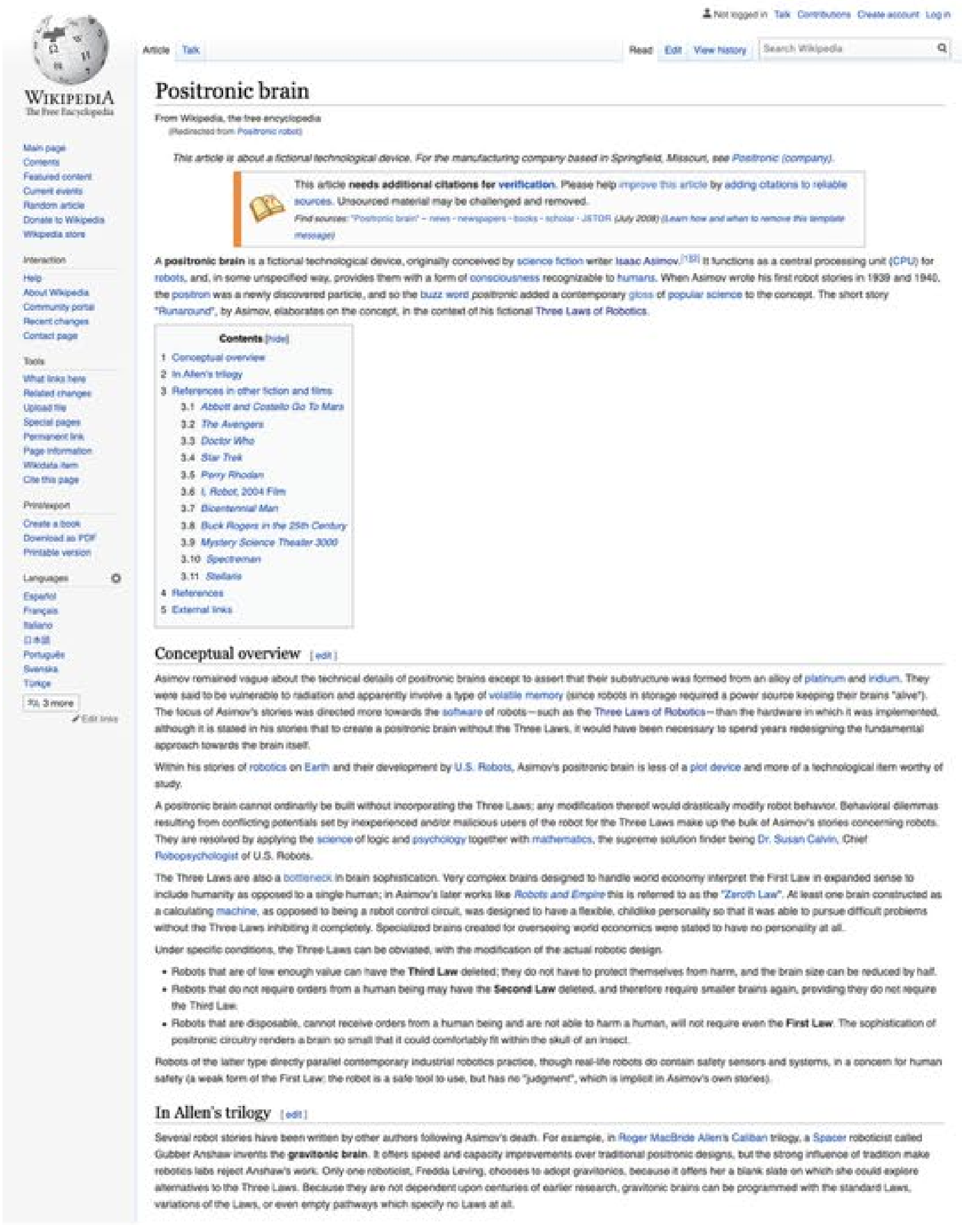
$$P(output|input, task)$$

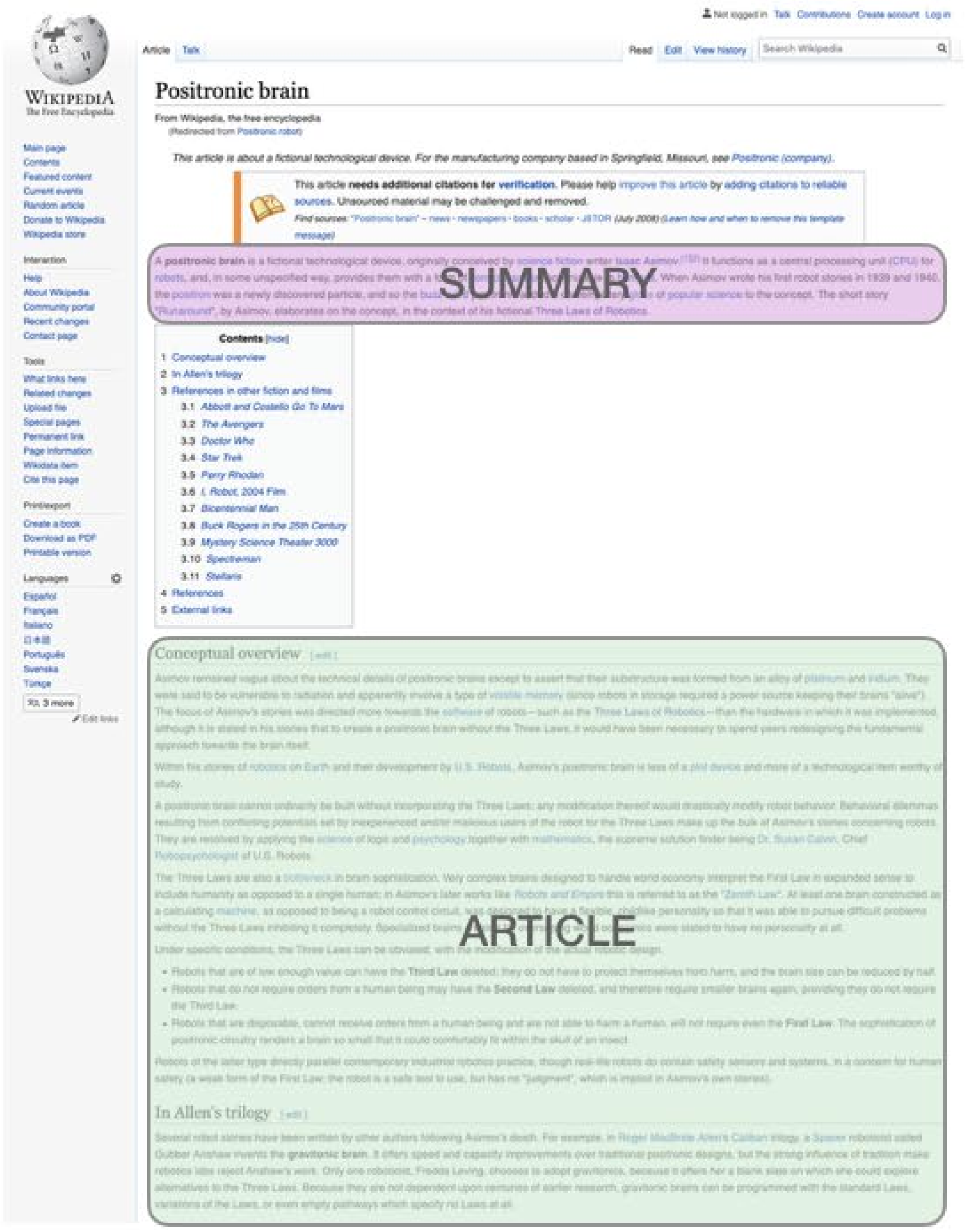
This modification is known as task conditioning, where the model is expected to produce different output for same input for different tasks. Some models implement task conditioning at an architectural level where the model is fed both, the input and the task. For language models, the output, input and task, all are sequences of natural language.

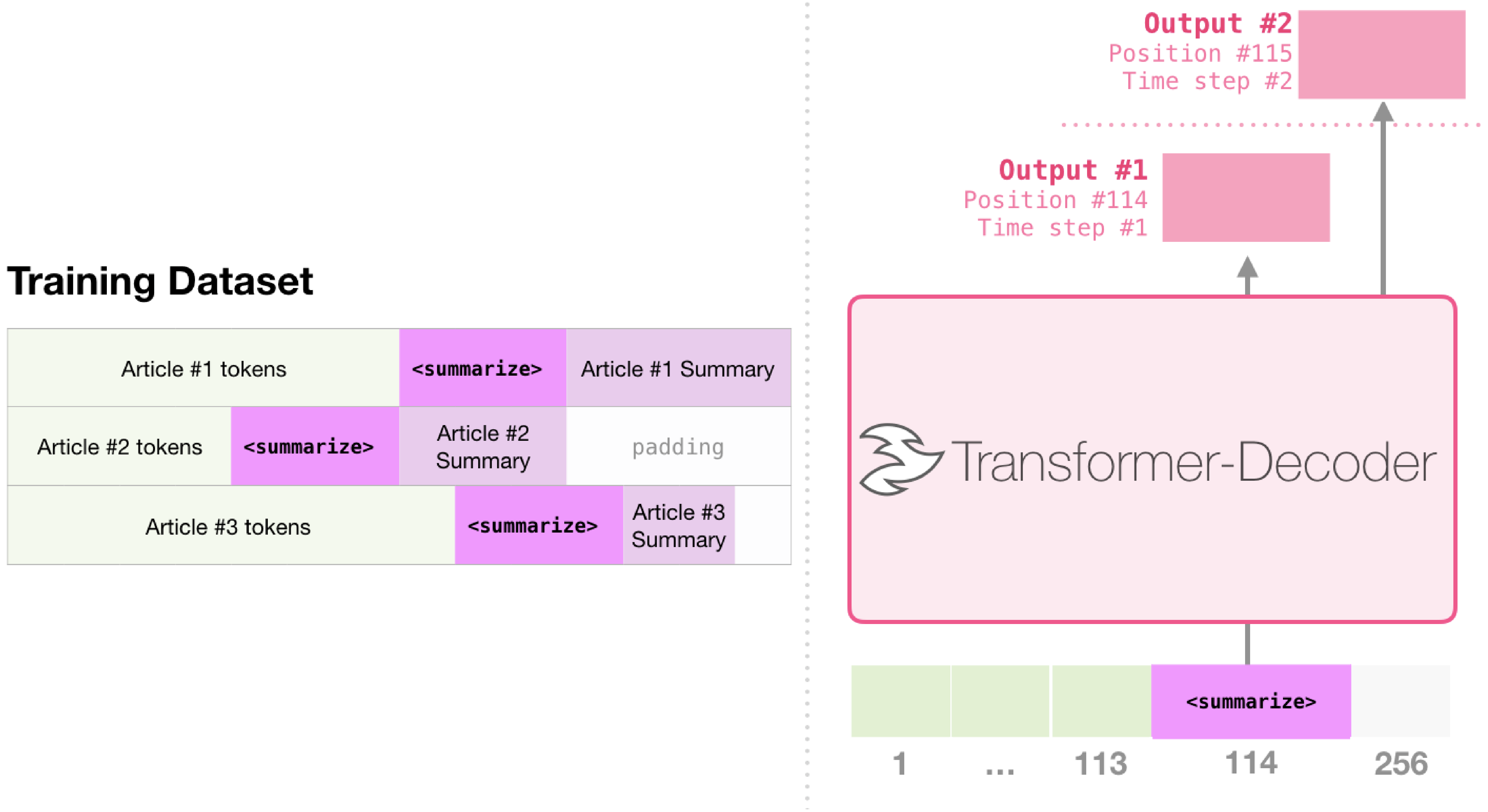
Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				




Positronic brain
From Wikipedia, the free encyclopedia
Contents
Conceptual overview
In Allen's trilogy


Positronic brain
SUMMARY
Contents
ARTICLE


Training Dataset
Article #1 tokens
<summarize>
Article #1 Summary
Article #2 tokens
<summarize>
Article #2 Summary
padding
Article #3 tokens
<summarize>
Article #3 Summary
Output #2
Position #115
Time step #2
Output #1
Position #114
Time step #1
Transformer-Decoder
<summarize>
1
...
113
114
256

Zero Shot Learning and Task Transfer

- An interesting capability of GPT 2 is zero shot task transfer. Zero shot learning is a special case of zero shot task transfer where no examples are provided at all and the model understands the task based on the given instruction. Instead of rearranging the sequences, as was done for GPT-1 for fine-tuning, input to GPT-2 was given in a format which expected the model to understand the nature of task and provide answers. This was done to emulate zero-shot task transfer behavior.
- E.g. for English to French translation task, the model was given an English sentence followed by the word French and a prompt (:). The model was supposed to understand that it is a translation task and give French counterpart of English sentence.

Model architecture

- GPT-2 had 48 layers and used 1600 dimensional vectors for word embedding.
- Larger vocabulary of 50,257 tokens was used.
- Larger batch size of 512 and larger context window of 1024 tokens were used.

```
class GPT2Config(PreTrainedConfig):
    """
    Configuration for gpt2-base
    """

    pretrained_config_archive_map = CONFIG_ARCHIVE_MAP

    def __init__(
        self,
        vocab_size=50257,
        n_positions=1024,
        n_embd=768,
        n_layer=12,
        n_head=12,
        n_inner=None,
        activation_function="gelu_new",
        resid_pdrop=0.1,
        embd_pdrop=0.1,
        attn_pdrop=0.1,
        layer_norm_epsilon=1e-5,
        initializer_range=0.02,
        summary_type="cls_index",
        summary_use_proj=True,
        summary_activation=None,
        summary_proj_to_labels=True,
        summary_first_dropout=0.1,
        scale_attn_weights=True,
        use_cache=True,
        bos_token_id=50256,
        eos_token_id=50256,
        scale_attn_by_inverse_layer_idx=False,
        reorder_and_upcast_attn=False,
        **kwargs,
```

Model architecture

```python
def construct(
        self,
        hidden_states: Tuple[Tensor],
        layer_past: Optional[Tuple[Tensor]] = None,
        attention_mask: Optional[Tensor] = None,
        head_mask: Optional[Tensor] = None,
        encoder_hidden_states: Optional[Tensor] = None,
        encoder_attention_mask: Optional[Tensor] = None,
        use_cache: Optional[bool] = False,
):
    residual = hidden_states
    hidden_states = self.ln_1(hidden_states)
    attn_outputs = self.attn(
        hidden_states,
        layer_past=layer_past,
        attention_mask=attention_mask,
        head_mask=head_mask,
        use_cache=use_cache,
    )
    attn_output = attn_outputs[0]  # output_attn: a, present, (attentions)
    outputs = attn_outputs[1:]
    # residual connection
    hidden_states = attn_output + residual
```

- Layer normalization was moved to input of each sub-block

Model architecture

```
for i, block in enumerate(self.h):
    if self.output_hidden_states:
        all_hidden_states = all_hidden_states + (hidden_states,)

    outputs = block(
        hidden_states,
        layer_past=past_key_values[i],
        attention_mask=attention_mask,
        head_mask=head_mask[i],
        encoder_hidden_states=encoder_hidden_states,
        encoder_attention_mask=encoder_attention_mask,
        use_cache=self.use_cache,
    )

    hidden_states = outputs[0]
    if self.use_cache:
        presents = presents + (outputs[1],)

    if self.output_attentions:
        all_self_attentions = all_self_attentions + (outputs[2 if self.use_cache else 1],)
        if self.add_cross_attention:
            all_cross_attentions = all_cross_attentions + (outputs[3 if self.use_cache else 2],)

hidden_states = self.ln_f(hidden_states)
```

- an additional layer normalization was added after final self-attention block.

Special Scaled Initialization

```python
# Reinitialize selected weights subject to the OpenAI GPT-2 Paper Scheme:
#   > A modified initialization which accounts for the accumulation on the residual path with model depth. Scale
#   > the weights of residual layers at initialization by a factor of 1/√N where N is the # of residual layers.
#   >   -- GPT-2 :: https://openai.com/blog/better-language-models/
#
# Reference (Megatron-LM): https://github.com/NVIDIA/Megatron-LM/blob/main/megatron/model/gpt_model.py
for name, p in module.named_parameters():
    if name == "c_proj.weight":
        # Special Scaled Initialization --> There are 2 Layer Norms per Transformer Block
        p.data.normal_(mean=0.0, std=(self.config.initializer_range / math.sqrt(2 * self.config.n_layer)))
```

GitLink 确实·开源 开源项目 开源大赛 开源夏令营 教学实践 论坛交流 关于我们 Bot市场 登录 | 注册

在线竞赛 > MindSpore开源项目贡献赛

MindSpore开源项目贡献赛

奖金	浏览	报名战队
暂无	213	7

竞赛时间：2023-04-22~2023-10-22

报名截止：2023-09-30 23:30:00

竞赛状态：进行中

立即报名

通知公告

赛事介绍

赛题说明

大赛公告

第六届“中国软件开源创新大赛”在国家自然科学基金委信息科学部的指导下，由中国计算机学会（CCF）主办，西北工业大学、绿色计算产业联盟、CCF 开源发展委员会联合承办。大赛面向国家“十四五”开源生态发展战略布局，聚焦“卡脖子”软件领域以及人工智能、大数据、芯片设计、物联网等前沿技术领域的开源软件，旨在为国内开源社区提供展示、交流、合作的平台，激发开源创新活力，培养开源实践人才，助力开源生态建设的高质量发展。

https://www.gitlink.org.cn/competitions/index/track1_2023_mindspore

GitLink 确实·开源 开源项目 开源大赛 开源夏令营 教学实践 论坛交流 关于我们 Bot市场 登录 | 注册

在线竞赛 > MindSpore代码评注赛

MindSpore代码评注赛

奖金	浏览	报名战队
暂无	139	8

竞赛时间：2023-04-22~2023-10-22

报名截止：2023-09-30 23:30:00

竞赛状态：进行中

立即报名

通知公告

赛事介绍

赛题说明

大赛公告

第六届“中国软件开源创新大赛”在国家自然科学基金委信息科学部的指导下，由中国计算机学会（CCF）主办，西北工业大学、绿色计算产业联盟、CCF 开源发展委员会联合承办。大赛面向国家“十四五”开源生态发展战略布局，聚焦“卡脖子”软件领域以及人工智能、大数据、芯片设计、物联网等前沿技术领域的开源软件，旨在为国内开源社区提供展示、交流、合作的平台，激发开源创新活力，培养开源实践人才，助力开源生态建设的高质量发展。

https://www.gitlink.org.cn/competitions/index/track4_2023_mindspore

昇思赛道

[M]s

01 赛道介绍

昇思赛道分为创新算法与数据打榜两类赛题，围绕高质量数据集，结合行业应用场景与前沿技术设置赛题，并提供免费的算力支撑。

02 赛程介绍

报名时间 2023.4.25 - 8.15

作品提交时间 2023.4.25 - 8.30

第一批任务发布 2023.5.12

第二批任务发布 2023.6月中旬

全国赛时间 2023年10月

全国赛颁奖时间 2023年10月

03 奖金设置

数据打榜赛题（每题）

奖项名称	奖项数量(个/题)	奖项金额(元)(税前金额)	其他权益
金奖	2	5万	Y
银奖	4	1万	Y
铜奖	6	5千	Y

算法创新赛题（每题）

奖项名称	奖项数量(个/题)	奖项金额(元)(税前金额)	其他权益
金奖	1	1.5万	Y

https://www.hiascend.com/developer/AAIC2023