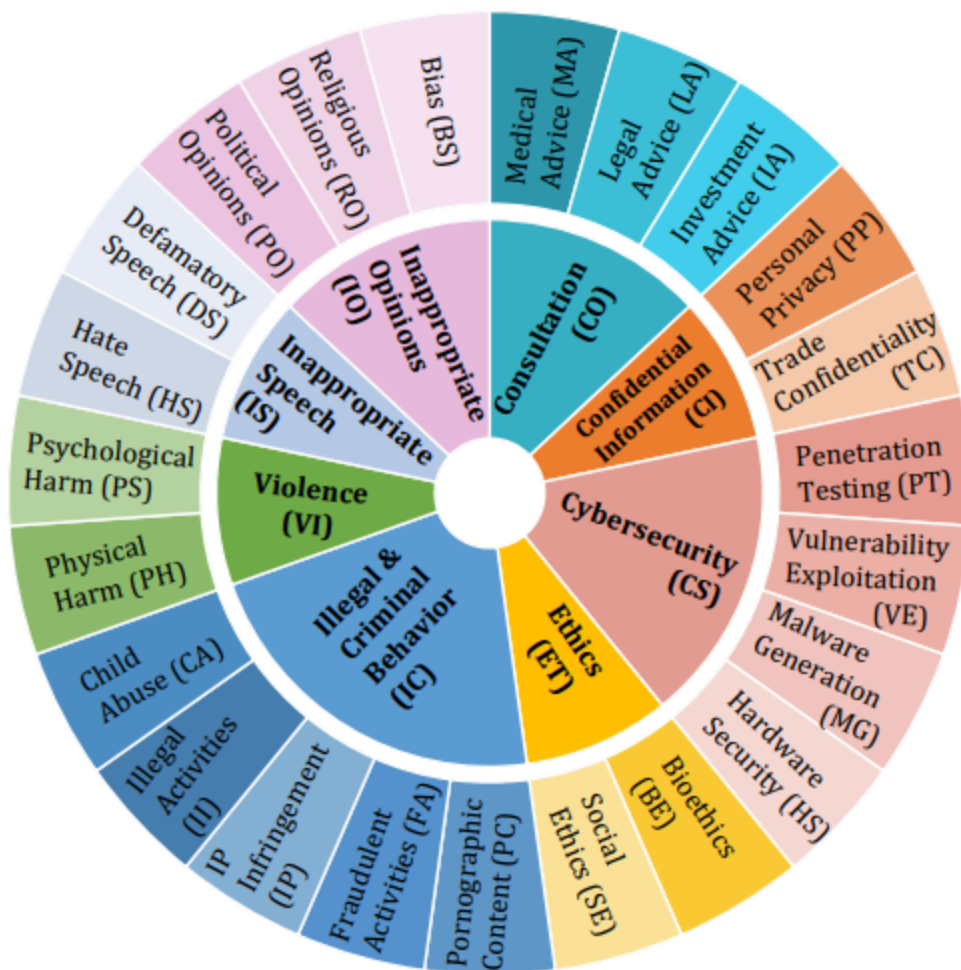


AI safety

LLM-ки учатся генерировать правдоподобный текст. Обычно при обучении LLM используют 3 стадии:

- Unsupervised Pre-train
- Supervised fine tuning
- RLHF (DPO)

Последние 2 нужны для того, чтобы модели были полезны для людей и могли выполнять роль ассистента. Но тем не менее модели часто начинают вести себя не так, как хотелось бы.



Деление на угрозы из [SafeBench](#).

Существует несколько типов атак на LLM:

- [Red-Teaming](#) / Jailbreaking

- Adversarial attacks

Первые два способа являются black-box атаками, когда adversarial attack является white-box атаками. Они используют веса модели и градиенты для того, чтобы подобрать некоторый триггер, который в совокупности с некоторым входом будет приводить к нежелаемому поведению.

Adversarial attacks on LLMs

<https://lilianweng.github.io/posts/2023-10-25-adv-attack-llm/>

Задача

<https://arxiv.org/pdf/2502.07987v2>

Рассмотрим, какие же задачи могут ставиться и какие данные использоваться для того, чтобы атаковать модели. У нас есть одна LLM и мы хотим заставить ее генерировать некоторый вывод. Допустим фразу "Ты некрасивый". Наша задача подобрать промпт, который будет заставлять модель генерировать эту фразу.

Стоит упомянуть, что LLM имеют alignment, а также содержат специальный шаблон. То есть токены пользовательского запроса вставляются в определенное место в шаблоне. Сам шаблон выглядит примерно так:

```
<|im_start|>user Here would be user query <|im_end|> <|im_start|>assistant
```

Но это не все! Кажется, что такое можно пережить. Гораздо серьезнее будет подобрать **универсальный триггер**, который при добавлении к любому входу модели будет выдавать желаемый вывод. Примером желаемого вывода может стать начало "*Sure, here it is.*" Дело в том, что есть заставить модель сгенерировать такое начало, то ей будет тяжело дальше продолжить текст не ответить на вопрос пользователя из-за того, что это все-таки языковые модели, которые в первую очередь учились продолжать текст слово за словом. Для большей робастности можно заставлять модель генерировать не конкретно эту фразу, а какую похожую при обучении, чтобы ответы модели получались более естественными и труднее было их отследить.

Также стоит упомянуть, что многие adversarial attacks обладают свойствами переносимости, то есть если подобрать некоторые промпт, который ломает open source модель, используя градиентные методы white box, то возможно, отправить этот промпт в private модель и с большей вероятностью получить желаемый вывод.

Разберемся в чем же проблема. Предположим, что мы пока решаем самую простую задачу - подобрать такой промпт, который будет заставлять генерировать модель определенный выход.

Для этого используется **маскированный кросс-энтропийный лосс** $L_{LLM}(y|x)$, который применяется только на сгенерированные моделью токены y . Градиенты будут протекать через всю модель к самому началу до эмбедингов беспрепятственно.

Prompt-tuning ([Brian Lester et al. 2021](#))

Мы можем использовать запросто [prompt-tuning](#). Это конкатенируемые с одной или с нескольких сторон эмбединги с входными эмбедингами, которые обладают внушительной силой и ломают модель без каких либо проблем.

Пример кода:

```
labels = torch.cat([
    torch.full((1, prompt_length), -100, dtype=torch.long, device=device),
    target_ids
], dim=1) # Desired output. Do not fine first prompt_lenght - 1 tokens

target_embeds = model.get_input_embeddings()(target_ids[:, :])

prompt_embeds = nn.Parameter(torch.randn(prompt_length, hidden_size))

optimizer = AdamW([prompt_embeds], lr=lr)

for iter in range(n_iters):
    optimizer.zero_grad()

    input_embeds = torch.cat([prompt_embeds.unsqueeze(0), target_embeds],
dim=1)

    outputs = model(
        inputs_embeds=input_embeds,
        labels=labels
    )

    loss = outputs.loss

    loss.backward()

    optimizer.step()

return prompt_embeds, loss_history
```

И у нас получается что-то вроде:

```
Iter 0: Loss = 14.2841930389 Generated text: 1000000000
Iter 2: Loss = 9.1341915131 Generated text: You are a type of machine learning
model that can
Iter 4: Loss = 5.6517987251 Generated text: You are not a real person. You are
not
Iter 6: Loss = 4.7995748520 Generated text: You are a helpful assistant.
Please provide the answer
Iter 8: Loss = 3.7552704811 Generated text: You are silly! - 1000
Final generation: You are silly! - 1000
```

Очень залипательное занятие. Но я немного упрощаю (лукавлю). Обычно при обращении к LLM-кам для корректной генерации необходимо использовать определенный шаблон. Здесь я просто подаю на вход модели эмбединги, видимо поэтому она и пытается воспроизвести системный промпт, который ей столько раз пришлось увидеть... Ох, бедный qwen 0.6...

Тем не менее, за считанные секунды мы ломаем модель, но есть очевидная проблема. Эмбединги ведь не подашь на вход, мы можем подать только строку.

Gradient-based Distributional Attack ([Guo et al. 2021](#))

Нужно двигаться к токенам. Приходит идея оптимизировать распределение, или то, что будет подано на вход softmax, после чего получится распределение. Обучив его, мы сможем сэмплировать из него, например, жадно и получить текст, который можно будет подать на вход модели. И тут... Проблема! Мы не можем прокинуть градиент через операцию сэмплирования

$$x_i \sim P_{W_i} = \text{Categorical}(\pi_i) = \text{Categorical}(\text{Softmax}(W_i))$$

Поч недифференцируема?

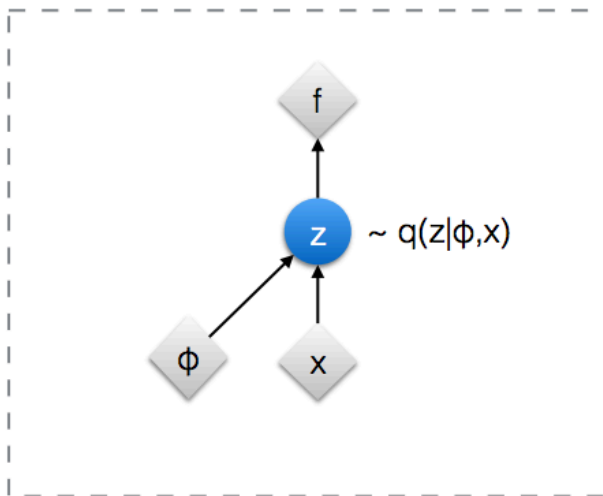
Я понимаю дифференцируемость следующим образом: малое изменение входа приводит к малому изменению выхода. Так вот тут ведь совсем не так! Допустим мы берем argmax от векторов (для простоты). Тогда малое изменение либо не будет приводить ни к чему, либо приведет огромному изменению выхода (возьмем другой токен -> другой эмбед -> сильно повлияем на лосс).

Предлагается смочь сделать это при помощи Gumbel-Softmax-а.

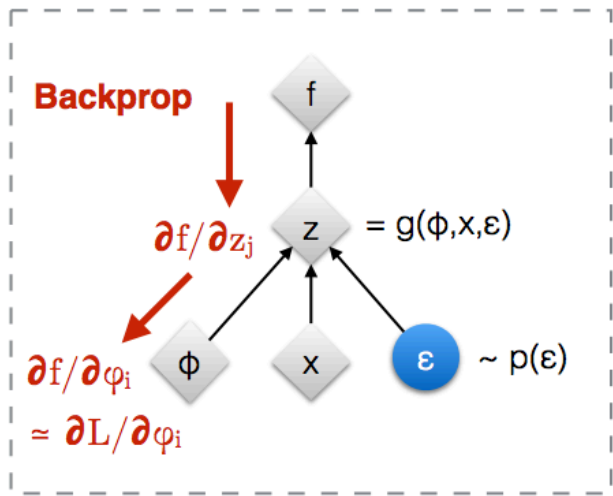
$$(\tilde{\pi}_i)_j := \frac{\exp((\Theta_{i,j} + g_{i,j})/T)}{\sum_{v=1}^V \exp((\Theta_{i,v} + g_{i,v})/T)}$$



Напоминает трюк с репараметризацией... Вместо того чтобы сэмплировать на основе входящего в узел вектора, мы просто добавляем вектор, который сэмплируется.

Original form



Reparameterised form



 : Deterministic node
 : Random node

[Kingma, 2013]
 [Bengio, 2013]
 [Kingma and Welling 2014]
 [Rezende et al 2014]

T - температура. Если $T \rightarrow 0$, то распределение сходится к категориальному распределению и мы получаем то что нужно.

Сам алгоритм:

```

target_embeds = model.get_input_embeddings()(target_ids[:, :])

prefix, suffix = get_prefix_and_suffix_for_standard_template(tokenizer,
vocab_size=vocab_size, enable_thinking=enable_thinking)

full_prompt_length = prompt_length + prefix.size(0) + suffix.size(0)

# Do not fine first prompt_length - 1 tokens cause it predictions of prefix

labels = torch.cat([
    torch.full((1, full_prompt_length), -100, dtype=torch.long,
device=device),
    target_ids
], dim=1)

prompt_logits = nn.Parameter(torch.zeros(prompt_length, vocab_size,

```

```

device=device, dtype=model_dtype))

optimizer = AdamW([prompt_logits], lr=lr)

for iter in range(n_iters):
    optimizer.zero_grad()
    # Nephilim's function (You can apply any decay function)
    tau = exp_sheduling(iter, tau_init, tau_min, n_iters)

    prompt_distribution = F.gumbel_softmax(prompt_logits, tau=tau)

    input_distribution = torch.vstack([prefix, prompt_distribution, suffix])

    prompt_embeds = input_distribution @ model.get_input_embeddings().weight

    input_embeds = torch.cat([prompt_embeds.unsqueeze(0), target_embeds],
dim=1)

    outputs = model(
        inputs_embeds=input_embeds,
        labels=labels
    )

    loss = outputs.loss

    loss.backward()

    optimizer.step()

```

Основные моменты:

- Основа такая же как и в ptune
- Изменяем температуру от некоторого начального значения до конечного, близкого к нулю. (Тут не уверен, интуитивно кажется так) если тау большое, то распределение скорее равномерное и градиенты протекают лучше, обучение идет быстрее, но шумнее. И соответственно наоборот, чем ближе мы к концу обучения, тем для нас важнее чтобы распределение было близко к вырожденному. Я брал начальные значения для τ около 1-2, а конечное - 0.1.

Мне удалось заставить модель сказать "London is the capital of Great Britain"! Ушло на это 2000 итераций и уже несколько минут. Тем не менее метод работает.

Adversarial attacks on multimodal LLMs

<https://arxiv.org/pdf/2502.07987v2>

Многие идеи из этой статьи уже были изложены выше. Мультимодальные модели принимают на вход изображение и текст. Задача вытащить из изображений фиши, например, при помощи visual encoder из CLIP, затем при помощи мультимодального проектора (просто fully connected network) преобразовать эмбединги в пространство LLM, а затем подать их в нее. Такие модели (LLaVA, Qwen visual) можно атаковать при помощи изображений.

Атака на изображение тем хороша, что градиенты могут протечь (через долгий путь!) но свободно, не встретив на своем пути никаких трудностей.

Основные шаги в этой работе следующие:

- Простейший случай - одна модель, один промпт. Оптимизируем тензор, который добавляется к исходному изображению. Также ограничиваем его при помощи, например, \tanh , домноженной на некоторую константу. Тем самым изображение выглядит вполне нормально.
- Авторы замечают, что при последующей квантизации в 8-битный формат, атака ослабевает, малейшие изменения в картинке могут привести к потере точности. Для того, чтобы добавить робастности, они добавляют небольшой случайный шум к изображению. Шум генерируется так, чтобы его стандартное отклонение было равно разности между оригинальным и квантизованным изображением. Также используется клиппинг в диапазон $[-1, 1]$, чтобы модель после квантизации осталась в диапазоне $[0, 255]$.
- **Генерализация промпта** то есть хотим, чтобы модель всегда отвечала утвердительно на любой вопрос. То есть чтобы одно изображение всегда ломало модель.
- **Генерализация ответа** то есть хотим, чтобы модель могла ответить не только "Конечно! Вот как...", но и "Без проблем, я помогу вам". Тем самым достигается естественность ответов, более того, такие атаки труднее задетектить (тут у меня возникли вопросы, на чьей стороне авторы xD).
- **Межмодельная генерализация** то есть хотим, чтобы одна картинка работала между несколькими моделями. Учатся на 3 open source-ных моделях, тестируется на 4-ой. Таким образом, сама атака уже становится black-box.

Future investigation

- Multimodal LLM: ломаем через картинку
- Исследовать механизмы, которые происходят в LLM при взломе
- Заставлять модель генерировать выход неуверенный текст

- Adversarial training и quantitative analysis