Diffusion Language Model

Zhiping (Patricia) Xiao
UCLA SCAI Lab Course Reading Group Fall 2022
Outline

Introduction
   References
   Background

Diffusion-LM
   Motivation
   Background
   Architecture
   Noise Schedule
   Training Objective
   Denoiser
   From Continuous to Discrete
   Controllable Text Generation
   Highlighted Results
Introduction
Diffusion-LM Improves Controllable Text Generation (NeurIPS’22)

- Build a Language Model eligible for fine-grained controllable text generation, by applying diffusion model on continuous latent space.
- Code: https://github.com/XiangLi1999/Diffusion-LM
Figure: Overview of different types of generative models.¹

¹lilianweng.github.io 2021 post on diffusion models
Given a sequence of discrete words $\mathbf{w} = [w_1, w_2, \ldots w_n]$.

A language model $p_{\text{lm}}(\mathbf{w})$ denotes the probability distribution over sequences of words.

A language model assigns a probability $p_{\text{lm}}(\mathbf{w}) = P(w_1, \ldots, w_m)$ to the whole sequence.
Controllable Text Generation is to compute $p(w|c)$, aiming at generate $w$ that satisfies the control target $c$, where $c$ is a control variable (e.g. syntax tree, sentiment, topic, politeness, to gender, persona).

Plug-and-Play Controllable Generation: aims to keep the LM frozen and steer its output using potential functions (e.g., classifiers). In this setting:

- $p_{lm}(w)$ pre-trained and frozen, encouraging fluent;
- For each task, train $p(c|w)$ on small amount of data, encourage $w$ to fulfill the control.
- Posterior $p(w|c)$ approximated from Bayes rule:

$$p(w|c) \propto p(c|w) \cdot p_{lm}(w)$$
Quick Recap: Bayes Rule

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]

In this case, \( p(c) \) can be regarded as a unchanged.

Therefore:

\[ p(w|c) \propto p(c|w) \cdot p_{lm}(w) \]
Diffusion-LM
Focusing on a language models (LMs) offering controllable generation for text without re-training.

Recent works succeed on controlling simple attributes e.g., sentiment, while little progress is made on complex, fine-grained controls (e.g., syntactic structure).

Solution: a new non-autoregressive language model based on continuous diffusions.

▶ continuous data domains: images, audio, etc. (enables efficient gradient-based controllable generation)
▶ previous text diffusion models: on discrete state spaces, defines a corruption process on discrete data (e.g., each token has some probability to be corrupted to an absorbing or random token).
Plug and play language models: A simple approach to controlled text generation (ICLR’20)

▶ It runs gradient ascent on an autoregressive LM’s hidden activations make the following tokens satisfy the control while maintaining fluency.

▶ Drawback 1: PPLM is based on autoregressive model, it can only generate left-to-right, thus can never repair previous errors.

▶ Drawback 2: Work well on simple attribute (e.g. topic) control tasks, fail on more complex control tasks (e.g. syntactic structure).
Recall that, the case of image:

\[ L_{DM} = \mathbb{E}_{x, \epsilon \sim \mathcal{N}(0,1), t} \left[ \| \epsilon - \epsilon_{\theta}(x_t, t) \|_2^2 \right] . \]

The case of text: to maximize \( \mathbb{E}_{x_0 \sim p_{\text{data}}} \left[ \log p_{\theta}(x_0) \right] \)

View as modeling latent variables of the data \( x_0 \in \mathbb{R}^{nd} \) (\( n \) is the seq length, \( d \) is dimension of t) as a Markov chain \( x_T, \ldots, x_0 \in \mathbb{R}^{nd} \) where \( x_T \) is a Gaussian. The initial state \( p_{\theta}(x_T) \approx \mathcal{N}(0, I) \), and noise to reduce at step \( t \):

\[ p_{\theta}(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \Sigma_{\theta}(x_t, t)) , \]

where \( \mu_{\theta} \) and \( \Sigma_{\theta} \) may be computed by a U-Net or a Tranformer. (Ablation Studies in Sec 7.4 and Appendix H)
Diffusion Models (DMs)

To train $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$:

- **Forward Process** $q$: incrementally adds Gaussian noise to data $x_0$, until at diffusion step $T$, samples $x_T$ are approximately Gaussian.

  \[
  q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I),
  \]

  where $\beta_t$ is the amount of noise added at step $t$. $q$ is pre-defined and contains no trainable parameter.

- **Reverse Process** $p_\theta$: reconstruct the data (i.e. $x_T \rightarrow \ldots \rightarrow x_0$), denoiser U-Net or Transformer.

- The **diffusion model** is trained to maximize the marginal likelihood of the data

  \[
  \mathbb{E}_{x_0 \sim p_{data}} \left[ \log p_\theta(x_0) \right]
  \]
Measurement of lm: Feed generated text to a teacher LM (i.e., a carefully fine-tuned GPT-2 model) and report the perplexity. This metric is called lm-score (lm), a lower lm-score indicates better sample quality.
Figure: Iteratively denoises a sequence of Gaussian vectors into word vectors, yielding a intermediate latent variables of decreasing noise level $x_T \ldots x_0$. For controllable generation: iteratively perform gradient updates on these continuous latents to optimize for fluency (parametrized by **Diffusion-LM**) and satisfy control requirements (parametrized by a **classifier**).
Figure: The forward and reverse diffusion processes. In addition to the original diffusion models, Diffusion-LM add a Markov transition between \( x_0 \) and \( w \), defining embedding (\( w \rightarrow x_0 \)) and rounding (\( x_0 \rightarrow w \)).
Autoregressive LMs: \( p_{lm}(w) \)

\[
p_{lm}(w) = p_{lm}(w_1) \prod_{i=2}^{n} p_{lm}(x_i|x_{<i}),
\]

where the next-token prediction \( p_{lm}(x_i|x_{<i}) \) is often parametrized by Transformer architecture.

Claim: Most large pre-trained LMs are left-to-right autoregressive. (e.g. GPT-3, PaLM)

- Fixed generation order (i.e., left to right) limits the flexibility of models.
- For more: https://huggingface.co/transformers/v3.1.0/model_summary.html
Diffusion LM: Non-Autoregressive

Non-autoregressive LMs:

Claim: Most of the existing models in this category are task-specific, such as machine translation and speech-to-text (e.g. CTC and Imputer, NAT))

- It has been shown that these models fail for language modeling in CoMMA paper.
- A Study of Non-autoregressive Model for Sequence Generation (ACL’20)

Claim: Diffusion-LM can condition on arbitrary classifiers, which could utilize complex, global properties of the sentence.
According to my personal opinion:

- Diffusion Models are often expensive.
  - Many steps needed for training and generation.
  - In this case, the denoiser – a U-Net or a Transformer.
- Non-autoregressive models generate a whole sequence at a time.
The noise scheduler / variance schedule is an important hyper-parameter to be determined.

- Elucidating the Design Space of Diffusion-Based Generative Models (NeurIPS’22)
- The scheduler decide how much noise we add at each step.
The noise scheduler / variance schedule is an important hyper-parameter to be determined.

- Elucidating the Design Space of Diffusion-Based Generative Models (NeurIPS’22)
- The scheduler decide how much noise we add at each step.

Problem: Standard noise schedules for continuous diffusion models are not robust for text data.
The noise scheduler / variance schedule is an important hyper-parameter to be determined.

- Elucidating the Design Space of Diffusion-Based Generative Models (NeurIPS’22)
- The scheduler decide how much noise we add at each step.

Problem: Standard noise schedules for continuous diffusion models are not robust for text data.

Hypothesis: discrete nature of text involves rounding, making the model insensitive to noise near $t = 0$.

Solution: introduce a new $\sqrt{t}$ noise schedule suits for text better.
Recall that during the forward process $q$ incrementally adds Gaussian noise to data $x_0$, until at diffusion step $T$, samples $x_T$ are approximately Gaussian.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_tI),$$

with $\beta_t$ to be scheduled, which is equivalent to:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon,$$

where $\epsilon \sim \mathcal{N}(0, I)$. In practice, they find re-parametrization helps with model training. By having $\alpha_t = \prod_{s=0}^{t}(1-\beta_s)$, we have the closed-form expression of $x_{t-1} = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon$.

We estimate $x_{t-1}$ as:

$$x_{t-1} = \sqrt{\alpha_t}f_\theta(x_t, t) + \sqrt{1-\alpha_t}\epsilon,$$

where $f_\theta(x_t, t)$ estimates $x_0$ directly.
Proposed in Auto-Encoding Variational Bayes (ICLR’14)

The combination of forward process $q$ and backward process $p_\theta$ can be seen as a variational auto-encoder (VAE), hence the variational lower bound (i.e. ELBO) can be used to minimize the negative log-likelihood with respect to ground truth data sample $x_0$. 
Proposed in Auto-Encoding Variational Bayes (ICLR’14)

- The combination of forward process $q$ and backward process $p_\theta$ can be seen as a variational auto-encoder (VAE), hence the variational lower bound (i.e. ELBO) can be used to minimize the negative log-likelihood with respect to ground truth data sample $x_0$.

- The ELBO in this case is the sum of losses at each step:

\[ L = L_0 + L_1 + \cdots + L_T \]
Proposed in Auto-Encoding Variational Bayes (ICLR’14)

- The combination of forward process $q$ and backward process $p_\theta$ can be seen as a variational auto-encoder (VAE), hence the variational lower bound (i.e. ELBO) can be used to minimize the negative log-likelihood with respect to ground truth data sample $x_0$.

- The ELBO in this case is the sum of losses at each step:

$$L = L_0 + L_1 + \cdots + L_T$$

- By the construction of $q$, we observe each $L_t$ ($t = 1, 2, \ldots T$) is the KL divergence between 2 Gaussian distributions.

$$\|\epsilon - \epsilon_\theta(x_t, t)\|^2 = \|\epsilon - \epsilon_\theta(\sqrt{\alpha_t}x_0 + \sqrt{(1 - \alpha_t)}\epsilon, t)\|^2$$
(*) The Re-Parametrization

\[ q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t}x_{t-1}, \beta_t I) \]

Note that: sum of Gaussians is still Gaussian. As is shown by Deep Unsupervised Learning using Nonequilibrium Thermodynamics (ICML’15):

- It means that we do not have to apply \( q \) repeatedly to sample \( x_t \) from \( x_0 \). Instead we have:

\[ q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_0, (1 - \alpha_t)I) , \]

with \( \alpha_t = 1 - \beta_t \) and \( \overline{\alpha}_t = \prod_{s=0}^{t} \alpha_s \)

- This then allows optimizing random terms of \( L_t \) of a randomly sampled \( t \) during training.

- This re-param trick turns the model from a step-\( t \) mean predictor to a total noise predictor.

- references: huggingface post, Lilian post.
For the noise schedule $\sqrt{1 - \alpha_t}$, sqrt schedule defines $\alpha_t$ as:

$$\alpha_t = 1 - \sqrt{\frac{t}{T}} + s,$$

where $s$ is a small constant that corresponds to the starting noise level.
For the noise schedule $\sqrt{1 - \alpha_t}$, sqrt schedule defines $\alpha_t$ as:

$$\alpha_t = 1 - \sqrt{\frac{t}{T}} + s,$$

where $s$ is a small constant that corresponds to the starting noise level.

- When $t = 0$, $\alpha_t = 1 - \sqrt{s}$, the $\sqrt{1 - \alpha_t}$ is $\sqrt[4]{s}$. 

Other Noise (/Variance) Schedule

Ref: https://huggingface.co/blog/annotated-diffusion

▶ Linear Schedule
  ▶ Example: original DDPM, set the forward process variances to constants increasing linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.
  ▶ Denoising Diffusion Probabilistic Models (NeurIPS’20)

▶ Cosine Schedule
  ▶ The equation (usually divided by $\alpha_0$ to normalize):

\[
\alpha_t = \left( \frac{\pi}{2} \times \frac{\cos\left(\frac{t}{T} + s\right)}{1 + s} \right)^2
\]

▶ Shown in Improved Denoising Diffusion Probabilistic Models (ICML’21) that cosine schedule achieves better results.
Figure: Noise Schedule. Visualizing the noise schedule $\sqrt{1-\alpha_t}$. Figure 5 in the paper.
Figure: Noise Schedule, lm score. *sqrt* is the best. Figure 6 Row 2 in the paper. No longer salient when applying $x_0$-parametrization trick.
The simplified version of the canonical objective of maximizing

\[ \mathbb{E}_{x_0 \sim p_{data}} [\log p_\theta(x_0)] \]

is the variational lower bound of \( \log p_\theta(x_0) \), \(^2\) simplified as (NO LONGER a valid lower bound but mathematically proved and empirically more stable):

\[ L_{\text{simple}}(x_0) = \sum_{t=1}^{T} \mathbb{E}_{q(x_t|x_0)} \left[ \| \mu_\theta(x_t, t) - \hat{\mu}(x_t, x_0) \|^2 \right], \]

where \( \hat{\mu}(x_t, x_0) \) is the mean of posterior \( q(x_{t-1}|x_0, x_t) \) which is a closed form Gaussian, and \( \mu_\theta(x_t, t) \) is the predicted mean of \( p_\theta(x_{t-1}|x_t) \), computed by a neural network (Transformer / U-Net).

Proof in Appendix E of Diffusion-LM. Uses the closed form solution of the KL divergence of Gaussian.

\(^2\)the canonical objective comes from ICML’15
The simplified version of the canonical objective of maximizing $\mathbb{E}_{x_0 \sim p_{\text{data}}} [\log p_\theta(x_0)]$ is:

$$L_{\text{simple}}(x_0) = \sum_{t=1}^{T} \mathbb{E}_{q(x_t|x_0)} \left[ \| \mu_\theta(x_t, t) - \hat{\mu}(x_t, x_0) \|^2 \right],$$

where $\hat{\mu}(x_t, x_0)$ is the mean of $q(x_{t-1}|x_0, x_t)$, and $\mu_\theta(x_t, t)$ is the predicted mean of $p_\theta(x_{t-1}|x_t)$. The end2end version objective:

$$L_{\text{e2e}}(w) = \mathbb{E}_{q_\phi(x_{0:T}|w)} \left[ L_{\text{simple}}(x_0) + \| \text{EMB}(w) - \mu_\theta(x_1, 1) \|^2 \ight. \left. - \log p_\theta(x_0|w) \right]$$
Diffusion LM: Denoiser

Two options:

- Transformer: BERT-base model. Performs better, became the default choice.
- U-Net: make only one change to the standard U-Net, turning all 2D-convolutional layers into 1D-convolutional layers so that the model handles text sequence instead of image matrix.

Set diffusion steps to be 2,000 in practice. When running multiple (i.e. 3), steps of optimization steps (i.e. Adagrad), for each diffusion steps, reduce 2,000 to 200.
Recap: BERT-base

Figure: The architecture is the same as BERT-base, but Diffusion LM trained it from scratch. (link to BERT paper)
Recap: U-Net

Figure: In Diffusion LM, we replace all the 2D conv layers with 1D conv layers. (link to U-Net paper)
Figure: Transformer v.s. U-Net. Note that the less lm score, the better. Therefore, Transformer is better.
Embedding:

- \( \text{EMB}(w_i) \in \mathbb{R}^d \) embed discrete word \( w_i \) into vector space.
- \( \text{EMB}(w) = [\text{EMB}(w_1), \ldots, \text{EMB}(w_n)] \in \mathbb{R}^{nd} \) denotes the embedding of a length-\( n \) sequence. A Markov transition is applied:

\[
q_\phi(x_0|w) = \mathcal{N}(\text{EMB}(w), \sigma_0 I),
\]

which is trained end-to-end with the other components.

Rounding: achieved by choosing the most probable word according to

\[
\arg \max p_\theta(w|x_0) = \prod_{i=1}^{n} p_\theta(w_i|x_i),
\]

where \( p_\theta(w_i|x_i) \) is a softmax distribution.
Figure: Learned v.s. Random Embeddings. Note that the less lm score, the better. Learned Embedding is better. Figure 6 Row 1 in the paper. Conclusion: Fixed pre-trained embedding or random Gaussian embeddings are worse than the embedding trained via an end-to-end framework.
Figure: Figure 3 in the paper: A t-SNE plot of the learned word embeddings. Each word is colored by its POS (part-of-speech).
Recall that rounding is achieved by choosing argmax of

\[ p_\theta(w|x_0) = \prod_{i=1}^{n} p_\theta(w_i|x_i), \]

where \( p_\theta(w_i|x_i) \) is a softmax distribution. Ideally, this rounding is sufficient to map back to \textit{discrete} text. The denoising steps should make \( x_0 \) lie exactly on the embedding of some word.
Recall that rounding is achieved by choosing argmax of

$$p_\theta(w|x_0) = \prod_{i=1}^{n} p_\theta(w_i|x_i),$$

where $p_\theta(w_i|x_i)$ is a softmax distribution. Ideally, this rounding is sufficient to map back to **discrete** text. The denoising steps should make $x_0$ lie exactly on the embedding of some word.

But the problem is that, empirically, the model fails to generate $x_0$ that commits to a single word.
One Explanation: not emphasizing single commit on $x_0$ enough.

$$L_{\text{simple}}(x_0) = \sum_{t=1}^{T} \mathbb{E}_{q(x_t|x_0)} \left[ \| \mu_\theta(x_t, t) - \hat{\mu}(x_t, x_0) \|^2 \right]$$

- The model $\mu_\theta(x_t, t)$ directly predicts the mean of $p_\theta(x_{t-1}|x_t)$ at every denoising step $t$.
- The constraint that $x_0$ has to commit to a single word embedding will only appear in the terms with $t$ near 0, and require careful tuning to emphasize those terms. (Appendix H)
- Quick Fix: $x_0$-parameterized model

$$L_{x_0-\text{simple}}^{e2e}(x_0) = \sum_{t=1}^{T} \mathbb{E}_{x_t} \| f_\theta(x_t, t) - x_0 \|^2.$$
**Figure:** Figure 4 right half in the paper. Parametrizing by $x_0$ consistently performs well, whereas parametrizing by $\epsilon$ works fine for small dimensions, but quickly collapses for larger dimensions.
x₀-parameterized model:

\[ \mathcal{L}^{e2e}_{x_0\text{-simple}}(x_0) = \sum_{t=1}^{T} \mathbb{E}_{x_t} \| f_\theta(x_t, t) - x_0 \|^2, \]

where our re-parameterized model \( f_\theta(x_t, t) \) learns \( x_0 \) directly.

In the decoding phase, same intuition could be used in clamping trick:

- Maps the predicted \( f_\theta(x_t, t) \) to its nearest word embedding sequence at every step.
- Forces the predicted vector to commit to a word for intermediate diffusion steps, making predictions more precise and reducing rounding errors.
Before clamping trick:

\[ x_{t-1} = \sqrt{\alpha_t} f_\theta(x_t, t) + \sqrt{1 - \alpha_t} \epsilon , \]

After clamping trick:

\[ x_{t-1} = \sqrt{\alpha_t} \cdot \text{Clamp}(f_\theta(x_t, t)) + \sqrt{1 - \alpha_t} \epsilon , \]
Controlling $x_{0:T}$ over $c$ is equivalent with decoding from the joint inference problem posterior:

$$p(x_{0:T}|c) = \prod_{t=1}^{T} p(x_{t-1}|x_t, c),$$

which can be decomposed to a sequence of control problems at each diffusion step:

$$p(x_{t-1}|x_t, c) \propto p(x_{t-1}|x_t) \cdot p(c|x_{t-1}, x_t)$$

And according to a Yang Song et al’s paper from ICLR’21 (Section 5), there are conditional independence assumptions that we can use to simplify:

$$p(c|x_{t-1}, x_t) = p(c|x_{t-1})$$
Therefore, for the $t$-th step, we run gradient update on $\mathbf{x}_{t-1}$:

$$
\nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) = \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t) + \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{c}|\mathbf{x}_{t-1}) ,
$$

where the two differentiable terms are parameterized by:

- $\log p(\mathbf{x}_{t-1}|\mathbf{x}_t)$: Diffusion LM (for fluency);
- $\log p(\mathbf{c}|\mathbf{x}_{t-1})$: an arbitrary neural network classifier (for control);

respectively.
Therefore, for the $t$-th step, we run gradient update on $\mathbf{x}_{t-1}$:

$$\nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) = \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t) + \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{c}|\mathbf{x}_{t-1}),$$

where the two differentiable terms are parameterized by:

- $\log p(\mathbf{x}_{t-1}|\mathbf{x}_t)$: Diffusion LM (for **fluency**);
- $\log p(\mathbf{c}|\mathbf{x}_{t-1})$: an arbitrary neural network classifier (for **control**);

respectively.

In practice, we add **fluency regularization** where $\lambda$ is a hyper-param:

$$\lambda \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t) + \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{c}|\mathbf{x}_{t-1}),$$
Steps:

1. Collect a set of samples $S$ drawn from the Diffusion-LM, instead of having one single best option.

2. Select the sample that achieves the minimum expected risk under a given loss function (e.g., negative BLEU score).

   \[
   \hat{w} = \arg \min_{w \in S} \frac{1}{|S|} \sum_{w' \in S} L(w, w')
   \]

3. A low quality sample would be dissimilar from the remaining samples, and penalized by the loss function.
<table>
<thead>
<tr>
<th>Method</th>
<th>Semantic Content</th>
<th>Parts-of-speech</th>
<th>Syntax Tree</th>
<th>Syntax Spans</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ctrl ↑</td>
<td>Im ↓</td>
<td>ctrl ↑</td>
<td>Im ↓</td>
<td>ctrl ↑</td>
</tr>
<tr>
<td>PPLM</td>
<td>9.9</td>
<td>5.32</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FUDGE</td>
<td>69.9</td>
<td>2.83</td>
<td>27.0</td>
<td>7.96</td>
<td>17.9</td>
</tr>
<tr>
<td>Diffusion-LM</td>
<td>81.2</td>
<td>2.55</td>
<td>90.0</td>
<td>5.16</td>
<td>86.0</td>
</tr>
<tr>
<td>FT-sample</td>
<td>72.5</td>
<td>2.87</td>
<td>89.5</td>
<td>4.72</td>
<td>64.8</td>
</tr>
<tr>
<td>FT-search</td>
<td>89.9</td>
<td>1.78</td>
<td>93.0</td>
<td>3.31</td>
<td>76.4</td>
</tr>
</tbody>
</table>

Table 2: Diffusion-LM achieves high success rate (ctrl ↑) and good fluency (Im ↓) across all 5 control tasks, outperforming the PPLM and FUDGE baselines. Our method even outperforms the fine-tuning oracle (FT) on controlling syntactic parse trees and spans.
Table 3: Qualitative examples from the Syntax Tree control. The syntactic parse tree is linearized by nested brackets representing the constituents, and we use the standard PTB syntactic categories. Tokens within each span are represented as *. We color failing spans red and bold the spans of interest that we discuss in §7.1.

FT refers to fine-tuned GPT-2 without plug-and-play setting.