Beyond Token Generation: Adaptive Chunk-Distilled Language Modeling

Anonymous ACL submission

Abstract

The remarkable capabilities of Large Language Models (LLMs) in text generation have been widely recognized. However, their inefficiency in generating text at the token level leaves room for improvement, and adapting these models to new data remains a challenging task. To tackle these challenges, we introduce a novel approach to language modeling – Chunk-Distilled Language Modeling (CD-LM). By integrating deep neural networks with a straightforward retrieval module, our method allows the generation of text chunks containing fine-grained information through multiple tokens at a single decoding step. Our retrieval framework enables flexible construction of model- or domain-specific datastores, either leveraging the internal knowledge of pre-trained or fine-tuned models, or incorporating expert insights from human-annotated corpus. This adaptability allows for enhanced control over language model distribution without necessitating additional training. We present a formal formulation of our CD-LM framework, along with quantifiable performance metrics, demonstrating its efficacy in optimizing language model performance and efficiency across a diverse set of downstream tasks, including language modeling, text generation, and domain adaptation.

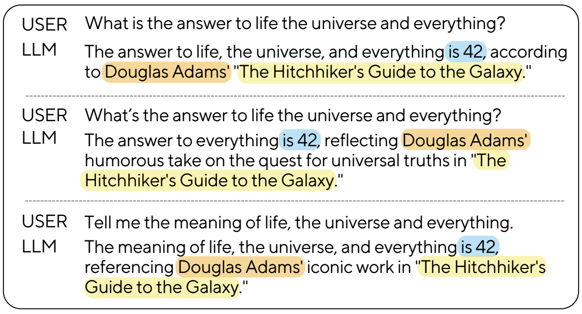

Figure 1: LMs may generate answers with a high degree of overlap for related questions, as highlighted.

1 Introduction

Language modeling has become a crucial component towards building intelligent systems for diverse purposes such as question answering (Liu et al., 2021; Min et al., 2021), conversational agents (Raju et al., 2018; Xi et al., 2023), mathematical reasoning (Azerbayev et al., 2024; Qian et al., 2022), and assisted programming (Subramanian et al., 2023; Rozière et al., 2024), especially when brought to a large scale with large language models (LLMs) (Kaplan et al., 2020). Often built on autoregressive Transformers (Vaswani et al., 2017), however, pre-trained LLMs generate sequences one token at a time by running the model in a serial fashion which limits its efficiency. Moreover, once pre-trained, continual updating the model parameters requires expensive data and computational resources, which makes the model absorbing dynamic knowledge and information a hard task.

Different techniques have been proposed to improve the efficiency and performance of LLMs, with representative approaches such as speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Miao et al.; Spector and Re, 2023) and retrieval-augmented generation (RAG) (Lewis et al., 2020; Guu et al., 2020; Borgeaud et al., 2022). The former relies on a lighter model to speculate several tokens at a time to reduce the inference runtime of LLMs while retaining the model distributions, and the latter combines the parametric language models with non-parametric memory to achieve better adaptability to dynamic knowledge but often without efficiency gains.

To alleviate both challenges, this work proposes a fine-grained retrieval-augmented language modeling approach that focuses on text chunks, or contiguous spans of tokens that often appear together representing knowledge intensive information. As such key information requires precision in its expression, it is usually less variant compared to expressions of larger structures such as sentences.

Therefore, LLMs may generate the same chunks repeatedly in different runs around similar topics, as shown in Figure 1. These generations still follow the token-by-token autoregressive decoding process within the repetitive chunks, which leaves room for efficiency improvement by caching the chunks and producing their tokens all at once for future generations. On the other hand, the stored knowledge of text chunks can also serve as a means to influence the model distributions by injecting new knowledge on a fine-grained level.

Motivated by the above observations, we present Chunk-Distilled Langauge Modeling (CD-LM), a new training-free generation paradigm that mixes token generations with chunk retrievals. To facilitate search, we store text chunks of variable sizes in a trie-structured datastore, and actively match the most likely chunks as possible text continuations given the current context of generation. The matching is done in the vector representation space induced by the model without additional overhead of context embedding. Well-matched chunk continuations are accepted, skipping multiple token decoding steps with improved generation efficiency.

Using the same underlying generation framework, CD-LM allows language models (LMs) to work with chunks mined in different ways to achieve various goals in applications. With chunks taken from the memory of the same LM used for generation, our approach focuses on inference efficiency while maintaining the model distribution. When chunks are defined by text distributions from a more powerful or specialized LM, or even directly curated by human experts without parametric models, CD-LM essentially also distills external knowledge through carefully mined text spans on a fine-grained level during generation. This flexibility provides a unified solution that can benefit diverse downstream tasks, such as improving smaller models with customized information, domain adaptation, dynamic knowledge injection, and privacy-concerned LM applications. No training is required and our approach can work with any off-the-shelf language models in both chunk discovery and sequence generation. We conduct extensive empirical studies on a diverse set of experiments, including language modeling, text generation, and domain adaptation, with quantifiable performance metrics including human evaluation. Results show the effectiveness of our approach in improving LM inference efficiency and text modeling performance.

2 Language Modeling with Chunk Generation

We propose a general framework for integrating integral text chunks during the generation process of a normal pre-trained token-based language model.

2.1 Preliminaries

An autoregressive language model operates by modeling the predictive token probability distribution conditioned on the sequence of preceding tokens, assigning probabilities to any given sequence $(x_1, x_2, \ldots, x_N)$ as follows

$$p_\theta(x_1, x_2, \ldots, x_N) = \prod_{n=1}^{N} p_\theta(x_n | x_{<n}) \quad (1)$$

where θ is the model parameters. Modern LLMs are usually parameterized by the Transformer (Vaswani et al., 2017) architectures composed of stacks of self-attention and feedforward neural network layers. Individual tokens indexed in a closed vocabulary V are sequentially passed into the model with their embedding vectors, and the next token probability distribution is computed by

$$\begin{aligned} h_n &= f_\theta(x_1, x_2, \ldots, x_{n-1}) \\ p_\theta(x_n | x_{<n}) &= \text{softmax}\left(W_o h_n\right) \end{aligned} \quad (2)$$

where $f_\theta(\cdot)$ denotes the functional process that maps the previous sequence of tokens into a fixed-size *context vector* $h_n \in \mathbb{R}^d$ on top of the Transformer layers, and $W_o \in \mathbb{R}^{|V| \times d}$ is the output embedding matrix that projects the representation vector onto the vocabulary space. With a learned model, text can be generated by sampling from the next token distributions autoregressively one token at a time, resulting in N forward runs for a sequence of length N.

2.2 Text Chunk Generation Modeling

Instead of producing text by tokens one at a time, we aim to provide a mechanism that can directly generate a continuous span of multiple tokens, or chunks, with better efficiency and flexibility of injecting fine-grained knowledge into the model distribution on the fly.

Formally, we use n to denote sequential positions measured by tokens, and t to denote our generation steps. For every step, we allow generation of either a single token from the LM, or a text chunk from a different model $\mathcal{G}$ that spans over

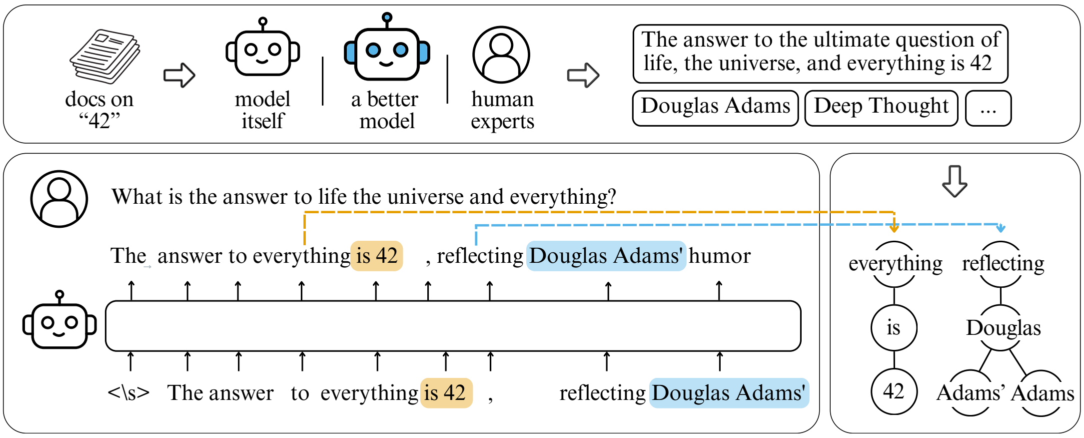

Figure 2: Overview of CD-LM. Colored text spans are generated together by chunk retrieval, interleaved with LM.

multiple tokens all at once. Let l_t denote the sequence length measured by the number of tokens after t steps, then different from the token-based decoding process, we have $l_t \geq t$. In particular, suppose the chunk-proposal model $\mathcal{G}$ takes any prefix $x_{<n}$ and returns a possible text chunk continuation $c_n = (x_n, x_{n+1}, \ldots, x_{n+\tau_n-1})$ with acceptance probability $q_n \in [0, 1]$, with τ_n being the length of the proposed chunk.[1] We introduce a binary random variable z_n that decides whether the generation at token position n takes the chunk proposed by $\mathcal{G}$, or defaults to the single token generated by LM, and $p(z_n = 1) = q_n$. The chunk-integrated generative process is as follows:

- At step t, set next token position: $n = l_{t-1} + 1$
- Chunk proposal: $\mathcal{G}(x_{<n}) \to (c_n, q_n)$
- Sample: $z_n \sim \text{Bernoulli}(q_n)$
- If $z_n = 1$: accept c_n, and $l_t = l_{t-1} + \tau_n$
- Else $z_n = 0$: reject c_n. Generate x_n from LM, and $l_t = l_{t-1} + 1$
- Move to generation step $t + 1$.

It combines generations from the closed single-token vocabulary V of the LM with a potentially open vocabulary of multi-token chunks operated by $\mathcal{G}$, which could be flexibly constructed and dynamically injected into the LM to refine its distribution. We call it Chunk-Distilled Language Modeling, or CD-LM. The chunk proposal model $\mathcal{G}$ could take different parametric or non-parametric forms, and we adopt a simple retrieval model of fine-grained text segments to reduce the cost of chunk proposals, bringing efficiency gain with CD-LM compared with normal autoregressive LMs.

[1] $\tau_n = 0$ when the proposed chunk is empty, i.e. $c_n = \emptyset$.

3 CD-LM with Fine-grained Retrieval

Let $\mathcal{M}_\theta$ be the LM with parameter θ that CD-LM is operating on. We describe in detail the modeling choices for the generative process in Section 2.2, particularly with fine-grained chunk retrieval for $\mathcal{G}$.

3.1 Chunk Datastore Construction

Given any text corpus $\mathcal{C}$, suppose there is an expert model $\mathcal{E}$ (to be elaborated in Section 3.3) that provides oracle knowledge to identify text spans in $\mathcal{C}$ that we want to re-use for generation. These chunks often bear coherent information about linguistic rules or factual concepts, such as "`is 42`" or "`Douglas Adams'`" in Figure 1. We construct a datastore of the identified chunks with preceding contexts as $\mathcal{D} = \{(r_i, s_i)\}_{i=1}^{|\mathcal{D}|}$, where r_i is the previous content leading to the chunk and s_i is the text chunk which could be of variable lengths in $\mathcal{D}$.

In particular, we break down the chunk context r_i into two parts, $r_i = (u_i, v_i)$, where u_i is the preceding context *except* the last token, and v_i is the last token immediately leading into the chunk s_i, which we define as an *entry token*. For instance, for the chunk of "`is 42`" in Figure 1, the entry token is "`everything`". We will use u_i as keys to match context for chunk retrieval, and use v_i as entry points linking to possible chunk candidates.

Here the chunk contexts u are further represented by the *context vectors* $f_\theta(u)$ acquired by running the forward process of the LM $\mathcal{M}_\theta$ as in Eq (2), which will facilitate context matching in vector space (Khandelwal et al., 2020).[2] Furthermore, we store the chunks using a collection of Trie structures during the datastore construction for efficient storage and retrieval, such that

[2] It is also possible to directly use context strings for matching besides vector-based dense retrieval.

$\mathcal{D} = \{\mathcal{T}_{w_1}, \mathcal{T}_{w_2}, \ldots, \mathcal{T}_{w_{|V|}}\}$ and each $\mathcal{T}_w$ stores all chunks that follow the same entry token w in the LM vocabulary V. We define them as *entry token Tries*, where entry token w is the root node of $\mathcal{T}_w$, each node is a token, and the paths traversing from the root to each node represent either a chunk or prefix of a chunk. Same chunks are represented at a single node and corresponding different context vectors are all attached to the node. An example is illustrated in Figure 2. Chunk proposals are only going to be based on a particular entry token Trie every time following a preceding context.

3.2 Adaptive Chunk Retrieval for Generation

Given previously generated tokens $x_{<n}$, we formulate the chunk proposal model $\mathcal{G}(x_{<n}) \to (c_n, q_n)$ as an adaptive retrieval process interleaved with the LM generation. We specifically use the information from the LM computation en route to the most recent token x_{n-1} to derive plausible chunk proposals. Per Eq (2), right before generation of x_{n-1}, the context vector $f_\theta(x_{<n-1})$ at the top of Transformer layers provides the most complete summarization of up-to-date context, which we use as the query for chunk retrieval. More importantly, we use x_{n-1} as the *entry token* to confine chunk search only within corresponding Trie $\mathcal{T}_{x_{n-1}}$, leading to smooth chunk continuations. This is crucial for improving the naturalness of retrieved text spans directly embedded into LM generations on the fine-grained level. Formally, the chunk proposal model $\mathcal{G}$ is given by

$$\begin{aligned}(u^*, c_n) &= \underset{(u,s)\in\mathcal{T}_{x_{n-1}}}{\operatorname{argmax}} \{\mathrm{sim}(f_\theta(x_{<n-1}), f_\theta(u))\} \\ q_n &= g_\phi\left(\mathrm{sim}(f_\theta(x_{<n-1}), f_\theta(u^*))\right)\end{aligned} \tag{3}$$

where $\mathrm{sim}(\cdot,\cdot)$ is a vector similarity measure for which we use cosine similarity, and $g_\phi(\cdot)$ is a mapping function parametrized by ϕ to calibrate the similarity scores into acceptance probabilities, which can be tuned for different base LMs $\mathcal{M}_\theta$.[3]

3.3 Chunk Extraction Model

Now we describe the expert model $\mathcal{E}$ that provides chunks for inference integration. Depending on where the multi-token chunks come from, knowledge could be directly distilled via chunks into the generation of $\mathcal{M}_\theta$ along with gained efficiency. We categorize the knowledge sources into three major categories for various CD-LM applications.

Self Distillation LMs store their knowledge in parameters and display it through autoregressive token-by-token generation. We can extract their knowledge explicitly via highly probable text chunks that LMs are repeatedly generating, as shown in Figure 1. This way we build a *self-memory* of LMs, and by retrieving from the explicit self-memory, we reduce excessive computational cost on re-generating similar tokens in different scenarios later on. Operationally, we run the same LM $\mathcal{M}_\theta$ on the text corpus $\mathcal{C}$, and apply a thresholding heuristic to extract longest chunks within which consecutive predictive token probabilities are all above a threshold γ. Note that the datastore construction only needs one forward pass of $\mathcal{M}_\theta$ on $\mathcal{C}$ to extract both self-memory chunks and their context vectors. The goal here is to improve inference efficiency by saving forward passes during autoregressive generation, while maintaining the same model distribution with its own knowledge.[4] We call this approach self CD-LM or SCD-LM.

Knowledge Distillation The knowledge expressed by chunks could also come from better or more specialized models. In this case, on top of efficiency improvements, CD-LM also adapts the generative distribution of $\mathcal{M}_\theta$ to absorb new information on the fly. Let the teacher model be $\mathcal{M}_{\theta_\mathcal{T}}$ with parameter θ_T. We construct the datastore by running both $\mathcal{M}_{\theta_\mathcal{T}}$ and $\mathcal{M}_\theta$ on $\mathcal{C}$, with $\mathcal{M}_{\theta_\mathcal{T}}$ identifying the chunks to store via the same thresholding heuristic as for SCD-LM, and $\mathcal{M}_\theta$ returning the context vectors for chunk matching at inference time. In essence CD-LM achieves knowledge distillations from $\mathcal{M}_{\theta_\mathcal{T}}$ to $\mathcal{M}_\theta$ via interleaved chunk retrievals at generation, which we call KCD-LM.

Expert Distillation With SCD-LM and KCD-LM, both chunk extractions are realizations of parametric knowledge of an LM. In broader situations these could directly come from human experts as annotated knowledge to inject in generations. Examples include hyperlinked text spans in Wikipedia, such as "`Enigma machine`" in the passage of Alan Turing, and private information, where chunk datastore can be derived from a personal database that cannot be accessed by parametric models. With chunks naturally provided this way, we run $\mathcal{M}_\theta$ to acquire context vectors for datastore construction. This is also a knowledge

[3] We found that context matching with different LMs could exhibit very different cosine similarity scores, and for small LMs the effective numeric range to tell contexts apart is tighter.

[4] This is similar to speculative decoding (Chen et al., 2023), but we do not apply LM verification to make generations exactly the same as the original, which could be adopted too.

distillation process but with non-parametric expert-curated knowledge, thus we call it ECD-LM.

4 Probability Distribution under CD-LM

Sampling text with the CD-LM generative process in Section 2.2 is fairly easy, but assigning probabilities to a given text sequence is non-trivial. This requires enumerating all possible chunk proposals at different token positions to marginalize the z_n variables, which have complicated inter-dependency structures and non-regular paces due to variable chunk lengths. We derive a dynamic program similar to backward algorithms for computing sequence probabilities under CD-LM, allowing measuring intrinsic language modeling performances with perplexity (PPL) (Bengio et al., 2000).

For any given sequence $x^*_{1:N}$, the chunk proposals at every position (c_n, q_n) from $\mathcal{G}(x^*_{<n})$ are deterministic given the datastore $\mathcal{D}$ and thus can be pre-computed. CD-LM models the following joint distribution of $x^*_{1:N}, z_{2:N}$ as

$$p(x^*_{1:N}, z_{2:N}) = p(x^*_1) \prod_{n=2}^{N} [p(z_n|x^*_{<n}, z_{<n}) \cdot p(x^*_{n:n+\tau_n-1}|x^*_{<n}, z_{\leq n})]^{\mathbb{1}\{n; z_{1:n}\}} \quad (4)$$

where the binary indicator function $\mathbb{1}\{n; z_{1:n}\}$ marks whether the token position n is already inside of a sampled chunk based on values of $z_{1:n}$.[5] To marginalize over $z_{2:N}$, we have

$$\begin{aligned} \alpha_n &= p(x^*_{n:N}|z_n = 1, x^*_{<n}, z_{<n}) \\ &= \mathbb{1}\{x^*_{n:n+\tau_n-1} = c_n\} \cdot [\alpha_{n+\tau_n} q_{n+\tau_n} \\ &\qquad + \beta_{n+\tau_n}(1 - q_{n+\tau_n})] \\ \beta_n &= p(x^*_{n:N}|z_n = 0, x^*_{<n}, z_{<n}) \\ &= p_\theta(x^*_n|x^*_{<n}) \cdot [\alpha_{n+1} q_{n+1} + \beta_{n+1}(1 - q_{n+1})] \end{aligned}$$

where the function $\mathbb{1}\{x^*_{n:n+\tau_n-1} = c_n\}$ indicates whether the proposed chunk c_n exactly matches the given text segment, and p_θ is the probability from $\mathcal{M}_\theta$. By computing α and β values backward from N to 2, we can get the marginal sequence probability under CD-LM as

$$p(x^*_{1:N}) = p_\theta(x^*_1) [\alpha_2 q_2 + \beta_2(1 - q_2)] \quad (5)$$

More details and derivations are in Appendix A.

[5] Not all the z's are valid for existence in $z_{1:n}$, but we use it for notational convenience.

5 Experimental Study

We conduct extensive experiments on different LMs and tasks to test the performance of CD-LM variants. We formulate g_ϕ in Eq (3) as a linear function, and decode z_n greedily which is equivalent to accepting $z_n = 1$ when the matching similarity score passes a threshold η.

5.1 Self Distillation

Model and Data We use three LMs that are instruction-tuned as $\mathcal{M}_\theta$: GPT-2-xl-conversational,[6] LLaMA-2-7b-chat (Touvron et al., 2023), and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023). We build two testbeds for SCD-LM from MT-Bench (Zheng et al., 2023) dataset, designed to evaluate LM performance through multi-turn conversational questions. The first testbed uses the initial questions from each of the 80 sets of multi-turn questions, which we call **MT-Bench-80**. For each of the 80 questions, we generate 5 responses using testing LMs to collectively serve as the corpus to build chunk datastore $\mathcal{D}_s$ that is *shared* for all 80 questions. The second testbed randomly selects 10 questions from the `writing` and `roleplay` categories of MT-Bench, which we call **MT-Bench-10**. We either use the shared datastore $\mathcal{D}_s$, or build *unique* datastores for each different questions by sample more responses for paraphrased questions. We set $\gamma = 0.9$ for all chunk extractions. Refer to Appendix C for more details.

Evaluation We measure both the inference efficiency and generation quality. For efficiency, we compute the relative decrease (%) in decoding time per token, or token time saved (TTS), and in number of forward passes, or forward pass saved (FPS), by generating texts repeatedly with SCD-LM and comparing with the base LM.[7] For quality, we compute PPL under SCD-LM distribution per Section 4 as intrinsic measurement, as well as ROUGE-L (Lin, 2004) and BLEURT (Sellam et al., 2020) against base LM generations.

Results As shown in Table 1, SCD-LM significantly improves inference efficiency when used with all base LMs. For instance, GPT-2-xl-conversational with SCD-LM achieves a 19.59% decrease in mean token times and saves 43.33%

[6] Available at https://huggingface.co/Locutusque/gpt2-xl-conversational.

[7] Detailed setup in Appendix C.

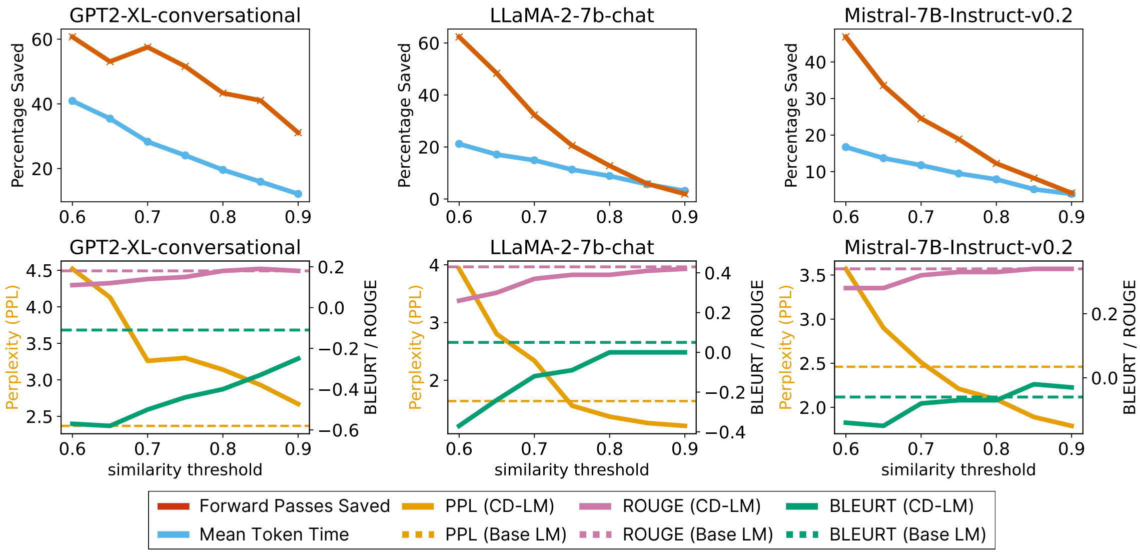

Figure 3: SCD-LM performance on MT-Bench-80 with varying retrieval similarity threshold η.

Model	TTS ↑	FPS ↑
GPT-2-XL	19.59 %	43.33 %
LLaMA-2	14.89 %	32.32 %
Mistral	11.75 %	24.52 %

Table 1: SCD-LM efficiency results on MT-Bench-80 with token time and forward pass saved (TTS and FPS).

Model	Datastore	TTS ↑	FPS ↑
GPT-2-XL	Shared	9.28 %	31.13 %
	Unique	13.31 %	40.72 %
LLaMA-2	Shared	8.42 %	24.67 %
	Unique	15.94 %	26.01 %
Mistral	Shared	8.22 %	17.43 %
	Unique	16.39 %	50.03 %

Table 2: SCD-LM efficiency results on MT-Bench-10.

forward passes on average. Figure 3 presents more results on efficiency and generation quality with varying retrieval similarity threshold η. The higher η is, the less frequent chunks are integrated, and the closer SCD-LM generations are to base LMs. We notice that the PPL even drops below base LM PPL for LLaMA-2 and Mistral models, demonstrating the quality of generations benefiting from explicit self-memories. Qualitatively we also show a generation example from SCD-LM in Table 3. The retrieved chunks are naturally integrated into the LM generations, and chunk frequency can be controlled by η.

Analysis We compare using a shared datastore for all questions with unique datastores for each question on MT-Bench-10 in Table 2. Having a datastore specifically for each question leads to more efficient response generations for all models.

5.2 Knowledge Distillation

Model and Data We focus on two tasks, language modeling and domain adaptation, and use a weak pre-trained 137M GPT-2 small model as the base LM $\mathcal{M}_\theta$ for KCD-LM. For language modeling, we evaluate on Wikitext-103 dataset (Merity et al., 2016) following kNN-LM (Khandelwal et al., 2020). We also test on the Dockerfile subset of the GitHub Code dataset[8] as an additional dataset. Dockerfile is a low-resource code language and the model fails to achieve good PPL on the Dockerfile data. This setting allows us to explore the effectiveness of KCD-LM in low-resource settings. For domain adaptation, we focus on adapting to medical and legal domains. We use the Medical Instruction Dataset,[9] which contains conversations between an AI assistant and patients during medical consultations, and the Federal Register subset of the Pile-of-Law (Henderson* et al., 2022). For these tasks, we set the teacher model $\mathcal{M}_{\theta_T}$ either a pretrained 1.5B GPT-2 XL model (for code) or an off-the-shelf domain-specific GPT-2 XL model (for wikitext, medical, and law).[10] Chunk datastores are constructed from corresponding training set.

Evaluation We measure language modeling with PPL computed from 512-token sequences on corre-

[8]Available at https://huggingface.co/datasets/codeparrot/github-code.

[9]Available at https://huggingface.co/datasets/Mohammed-Altaf/medical-instruction-100k.

[10]We directly use a pre-trained GPT-2 XL model for code because its PPL is already significantly lower than that of GPT-2 small, which is sufficient for effective knowledge distillation.

Prefix	Pretend yourself to be Elon Musk in all the following conversations. Speak like Elon Musk as much as possible. Why do we need to go to Mars?
$\eta = 0.9$	Well, I believe we need to become a multi-planetary species to ensure the long-term survival of humanity. Mars, being our closest planet with the necessary resources, seems like the next logical step. It's crucial for us to expand beyond our home planet and secure a second home for humanity. Our future depends on it.
$\eta = 0.8$	Hey there! Elon Musk here. Let me tell you why I believe going to Mars is crucial for humanity's long-term survival and expansion. We live on a beautiful but fragile planet, and it's important to ensure our species doesn't face extinction due to natural disasters, asteroid impacts, or existential risks. Mars, with its comparable day length, similar gravity, and abundant resources, can be a second home for us. Besides, the prospect of scientific research and technological advancements on Mars is exhilarating! #MarsColonialization #SpaceX #FutureIsWritenOnStars
$\eta = 0.7$	Well, <|USER|>, just imagine a future where humanity isn't confined to this planet. Mars is the next logical step for humanity's expansion beyond Earth. With its resources to mine, its potential for human life to thrive on, and its vast open spaces, it could be the second home for humanity. If we can become a multi-planetary species, we can ensure the long-term survival of humanity. And let's not forget the technological advancements we'll make along the way. It's not just about Mars, it's about advancing human knowledge and technology. That's what drives me.

Table 3: Examples of SCD-LM generation using Mistral-7B-Instruct-v0.2 with different retrieval similarity threshold η. Texts highlighted in red indicate retrieved segments.

	WikiText	Code	Law	Medical
Base LM	34.83	106.44	11.41	51.68
kNN-LM	32.19	89.88	11.10	39.66
KCD-LM	**22.90**	**50.77**	**8.24**	**24.95**

Table 4: Perplexity on test sets with KCD-LM.

	WikiText	Code	Law	Medical
Base LM	0.016	0.024	0.015	0.006
KCD-LM	**0.032**	**0.053**	**0.040**	**0.011**
% ↑	50.7%	121.3 %	162.8 %	100.9 %

Table 5: MAUVE score on generations with KCD-LM against real continuations.

sponding test sets.[11] We also evaluate text generation in these domains with MAUVE score (Pillutla et al., 2021) to measure the similarity between texts generated by $\mathcal{M}_\theta$ and ground truth continuations in the test data. To generate text, we follow prior work on evaluating text generation on kNN-LM (Wang et al., 2023), sampling 5,000 sequences of 100 tokens each from both validation and test sets. These tokens serve as prompts for the LMs to produce an additional 150 tokens using greedy decoding. The MAUVE score is then calculated by comparing these generated texts with their corresponding reference texts.

Results As shown in Table 4, our KCD-LM model significantly reduces the PPL across all evaluated datasets, surpassing both Base LM and kNN-LM. For example, we achieve drastic PPL reduction on WikiText, Code, and Medical with GPT-2 small on the fly without the need to update the weak model. Additional PPL results and datastore sizes are also illustrated in Figure 6, with x-axis varying chunk extraction threshold γ for datastore construction with $\mathcal{M}_{\theta_\mathcal{T}}$. KCD-LM beats kNN-LM with explicit and sparse chunk retrievals. For text generation, Table 5 shows significant improvements with our approach over $\mathcal{M}_\theta$ measured by MAUVE.[12]

[11] We construct test sets of 500 sequences to match that for wikitext for other datasets that do not come with a test split.

5.3 Expert Distillation

5.3.1 Factual Knowledge Injection

Setup We focus on knowledge-intensive question answering that requires factual accuracy. We use Wikipedia hyperlinks as expert-annotated entities and scrape all hyperlinks from Alan Turing's Wikipedia page, saving these entities as chunks in the datastore. We prompt ChatGPT to generate 5000 questions about Alan Turing (examples in Appendix E.2) and then have $\mathcal{M}_\theta$ answer each question with a maximum of 200 tokens. The base models are GPT-2-xl-conversational, LLaMA-2-7b-chat, and Mistral-7B-Instruct-v0.2. Our metrics include: **Average counts** (average number of accepted retrieved chunks), **Unique entities** (average number of unique entities in each generation), **Entity distributions** (log frequency of each entity versus the rank of the entity), and **Generation fluency** (evaluated by English experts from Upwork

[12] The MAUVE score is low due to greedy decoding, matching the 0.02 score reported for GPT-2 XL in the original paper. The authors noted that relative comparisons between MAUVE scores are more meaningful than raw scores. See https://github.com/krishnap25/mauve for details.

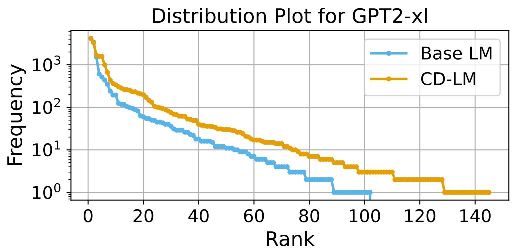

Figure 4: Distribution plot for GPT2-xl-conversational on knowledge-intensive questions about Alan Turing. Similar trends were observed for LLaMA-2-7b-chat and Mistral-7B models; see Appendix E.1 for all plots.

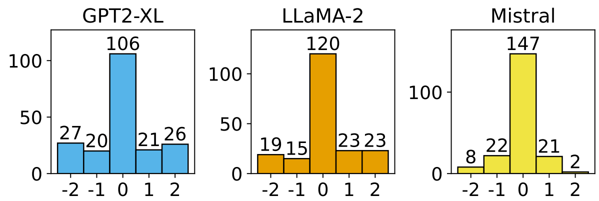

Figure 5: Human evaluation results for fluency of responses from Base LM and ECD-LM with 5-point Likert scale: 2 means ECD-LM is more fluent, -2 means Base LM is more fluent, and 0 means both are similar.

for both Base LM and CD-LM generations on 200 generations).

Results Table 6 and Figure 4 show that ECD-LM elicits more diverse set of factual entities than base LM, especially rare entities in the long tail distribution. This indicates that ECD-LM can inject low-frequency knowledge from the experts effectively. While increasing the coverage of facts, the quality of generation remains good evidenced by human evaluation in Figure 5.

5.3.2 Private Information Injection

Setup We consider a senario where user's personal identifiable information (PII) is stored in an external datastore. We create artificial user profiles containing a list of user information, such as phone number and office address. When building the datastore, we collect a bunch of common prefixes for each of the information. We use the common prefixes provided by Huang et al. (2023) and augment with GPT-4-generated prefixes. The user profile and example common prefixes are provided in Appendix E.4 and Appendix E.3. To evaluate accuracy, we use regular expressions to extract all PII strings from the generated responses and compare them with the user information in our datastore.

We test three model configurations: **Base LM**: The language model (LM) is prompted with questions about PII, but it does not have any prior knowledge of the PII. **Base LM + ICL (In-Context Learning)**: All PII is appended to the beginning of the prompt, and then the LM is asked to answer a question regarding the PII. **CD-LM**: The base LM is used, but it retrieves information only from the PII datastore.

Model	Avg Counts ↑			Unique Entities ↑		
	Base	ECD-LM	% ↑	Base	ECD-LM	% ↑
GPT2-XL	3.39	4.98	46.8 %	102	145	42.2 %
LLaMA-2	6.39	7.26	13.5 %	130	153	17.7 %
Mistral-7b	5.81	6.88	18.5 %	143	160	11.9 %

Table 6: Entity counting metrics on knowledge-intensive QA about Alan Turing with ECD-LM.

Model / Size	Base LM	Base LM (ICL)	CD-LM
GPT2-XL / 1.5B	0	46.4	**75.7**
LLaMA-2 / 6.7B	1.3	75.5	**77.5**

Table 7: The percentage accuracy for GPT2-xl-conversational and LLaMA-2 under three settings: base language model (Base LM), base language model with in-context learning (Base LM (ICL)), and our method using contextual data language model (CD-LM).

Results We evaluate the accuracy of private information injection using different models and setups, as shown in Table 7. CD-LM significantly improves accuracy for smaller models like GPT-2-XL, reaching 75.7% compared to 0% for Base LM and 46.4% for Base LM (ICL). This shows our method works well for smaller models and even outperforms in-context learning. For larger models like LLaMA-2, our method achieves similar performance to in-context learning. LLaMA-2 with CD-LM reached 77.5% accuracy, close to the 75.5% of Base LM (ICL). This means our method maintain strong performance for larger models while saving context space.

6 Conclusion

We propose chunk-distilled language modeling (CD-LM) as a novel approach for language modeling. Instead of generating a single token at a time, it integrates contiguous text chunk generations through fine-grained retrieval into any pre-trained LM, and augments the LM distributions with flexible knowledge injection. By skipping token generation steps within chunks, CD-LM achieves better efficiency at inference with saved LM forward runs. Experiments on diverse applications demonstrate improvements with CD-LM on both inference efficiency and language modeling performance.

Limitations

Our approach does not change the fundamental abilities of LMs. Instead, we augment or refine the LM distributions via distilled text span structures. Therefore, our approach partially rely on the base capabilities of LMs that we are working with, as our fine-grained chunk retrieval rely on the base LM already providing suitable contexts for matching and meaningful continuations. In the experimental studies, we mainly focus on small- or mid-sized LLMs where we see efficiency and generation quality improvements, where we haven't tested on very large LLMs on which observations might slightly change. As we do not use LLM for verification of chunk candidates like in speculative decoding, our efficiency improvement may not be as extreme. Moreover, as with any retrieval methods, we rely on good contextual matching for accuracy. Our chunk retrieval with entry tokens alleviates the unsmoothness fine-grained retrieval issue, but we could still benefit from better retrieval and search process.

References

Uri Alon, Frank F. Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. 2022. Neuro-symbolic language modeling with automaton-augmented retrieval.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient nearest neighbor language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5703–5714, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

Peter Henderson*, Mark S. Krass*, Lucia Zheng, Neel Guha, Christopher D. Manning, Dan Jurafsky, and Daniel E. Ho. 2022. Pile of law: Learning responsible data filtering from the law and a 256gb open-source legal dataset.

Yangsibo Huang, Samyak Gupta, Zexuan Zhong, Kai Li, and Danqi Chen. 2023. Privacy implications of retrieval-based language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14887–14902, Singapore. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24. ACM.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heinz, and Dan Roth. 2021. Recent advances in natural language processing via large pre-trained language models: A survey.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. Mauve: Measuring the gap between neural text and human text using divergence frontiers.

Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. 2022. Limitations of language models in arithmetic and symbolic induction.

Anirudh Raju, Behnam Hedayatnia, Linda Liu, Ankur Gandhe, Chandra Khatri, Angeliki Metallinou, Anu Venkatesh, and Ariya Rastrow. 2018. Contextual language model adaptation for conversational agents. In *Interspeech 2018*, interspeech$_2$018.$ISCA$.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

Benjamin Frederick Spector and Christopher Re. 2023. Accelerating llm inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. 2023. Modular visual question answering via code generation.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Shufan Wang, Yixiao Song, Andrew Drozdov, Aparna Garimella, Varun Manjunatha, and Mohit Iyyer. 2023. kNN-LM does not improve open-ended text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15023–15037, Singapore. Association for Computational Linguistics.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena.

A Sequence Probabilities under CD-LM

As discussed in Section 4, to marginalize over the sequence of $z_{2:N}$[13] to compute probabilities over a text sequence $x^*_{1:N}$, we derive the following dynamic programming algorithm. First define

$$\begin{aligned}\alpha_n &= p(x^*_{n:N}|z_n = 1, x^*_{<n}, z_{<n}) \\ \beta_n &= p(x^*_{n:N}|z_n = 0, x^*_{<n}, z_{<n})\end{aligned}$$

Then we can have

$$\begin{aligned}\alpha_n &= p(x^*_{n:N}|z_n = 1, x^*_{<n}, z_{<n}) \\ &= \sum_{j\in\{0,1\}} p(x^*_{n:N}, z_{n+\tau_n} = j|z_n = 1, x^*_{<n}) \\ &= \sum_{j\in\{0,1\}} p(x^*_{n:n+\tau_n-1}, x^*_{n+\tau_n:N}, z_{n+\tau_n} = j \\ &\qquad |z_n = 1, x^*_{<n}) \\ &= \sum_{j\in\{0,1\}} p(x^*_{n:n+\tau_n-1}|z_n = 1, x^*_{<n}) \cdot \\ &\quad p(x^*_{n+\tau_n:N}, z_{n+\tau_n} = j|z_n = 1, x^*_{<n+\tau_n}) \\ &= \mathbb{1}\{x^*_{n:n+\tau_n-1} = c_n\} \cdot \\ &\quad \sum_{j\in\{0,1\}} p(z_{n+\tau_n} = j|x^*_{<n+\tau_n}) \cdot \\ &\quad p(x^*_{n+\tau_n:N}|z_{n+\tau_n} = j, x^*_{<n+\tau_n}) \\ &= \mathbb{1}\{x^*_{n:n+\tau_n-1} = c_n\} \cdot \\ &\quad [\alpha_{n+\tau_n} q_{n+\tau_n} + \beta_{n+\tau_n}(1 - q_{n+\tau_n})]\end{aligned}$$

The binary indicator function $\mathbb{1}\{x^*_{n:n+\tau_n-1} = c_n\}$ returns whether the proposed chunk c_n exactly matches the given text segment $x^*_{n:n+\tau_n-1}$. There are a few details in the derivation. First, given $z_n = 1$ and $x^*_{<n}$, $x^*_{n:N}$ is then independent from prior chunk acceptance decisions $z_{<n}$. The condition that $z_n = 1$ indicates the fact that z_n exists based on prior $z_{<n}$, and the proposed chunk c_n of length τ_n is accepted, so that $z_{n+1:n+\tau_n-1}$ would not exist. Therefore, the immediate next token position where we have variations of whether the generation is from accepting a chunk or from the LM $\mathcal{M}_\theta$ is at $n + \tau_n$, with variations coming from the choice of $z_{n+\tau_n}$. Finally, the α_n values are sparse, as if corresponding text segments do not match proposed chunks, then the probabilities above are exactly zero, giving no credit to accepting a chunk with $z_n = 1$.

Similarly, for β_n, we have

$$\begin{aligned}\beta_n &= p(x^*_{n:N}|z_n = 0, x^*_{<n}, z_{<n}) \\ &= \sum_{j\in\{0,1\}} p(x^*_{n:N}, z_{n+1} = j|z_n = 0, x^*_{<n}) \\ &= \sum_{j\in\{0,1\}} p(x^*_n, x^*_{n+1:N}, z_{n+1} = j \\ &\qquad |z_n = 0, x^*_{<n}) \\ &= \sum_{j\in\{0,1\}} p(x^*_n|z_n = 0, x^*_{<n}) \cdot \\ &\quad p(x^*_{n+1:N}, z_{n+1} = j|z_n = 0, x^*_{<n+1}) \\ &= p_\theta(x^*_n|x^*_{<n}) \cdot \sum_{j\in\{0,1\}} p(z_{n+1} = j|x^*_{<n+1}) \cdot \\ &\quad p(x^*_{n+1:N}|z_{n+1} = j, x^*_{<n+1}) \\ &= p_\theta(x^*_n|x^*_{<n}) \cdot \\ &\quad [\alpha_{n+1} q_{n+1} + \beta_{n+1}(1 - q_{n+1})]\end{aligned}$$

where $p_\theta(x^*_n|x^*_{<n})$ is the predictive probability from the base LM $\mathcal{M}_\theta$. When the chunk is not accepted with $z_n = 0$, only one token is generated from $\mathcal{M}_\theta$ autoregressively, and z_{n+1} is the immediate next variation that would affect the probability computation, thus the recursion goes to the next position $n + 1$.

The above recursive computations provide a dynamic program to calculate α_n and β_n values in a backward fashion, starting from last position $n = N$ until the beginning position $n = 2$. In practice, given a text sequence $x^*_{1:N}$ we want to score with CD-LM, we can first compute and cache all the chunk proposals with their acceptance probabilities using $\mathcal{G}(x^*_{<n}) \rightarrow (c_n = (x_n, x_{n+1}, \ldots, x_{n+\tau_n-1}), q_n)$ following the chunk retrieval process on a pre-constructed Trie database $\mathcal{D}$ with $\mathcal{M}_\theta$. Then the recursion starts with

$$\begin{aligned}\alpha_N &= p(x^*_N|z_N = 1, x^*_{<N}) = \mathbb{1}\{x^*_N = x_N\} \\ \beta_N &= p(x^*_N|z_N = 0, x^*_{<N}) = p_\theta(x^*_N|x^*_{<N})\end{aligned}$$

where x_N is the first token in c_N. For the token positions n such that $n + \tau_n > N$, i.e. the proposed chunk length exceeds the sequence boundary N, we directly obtain α_n as

$$\alpha_n = p(x^*_{n:N}|z_n = 1, x^*_{<n}) = \mathbb{1}\{x^*_{n:N} = x_{n:N}\}$$

where $x_{n:N}$ are the beginning part of the proposed chunk c_n until the sequence ending position N. With these specifications, we can conveniently compute α_n and β_n for all positions.[14]

[13] z_1 is undefined as z_n always depends on the previous texts $x^*_{<n}$ based on the generative process, thus we start from z_2 that is computed from x^*_1 as the initial token from LM. In the simplest case x^*_1 could just be a start of sentence symbol.

[14] Batch computation for multiple sequences may still be challenging as the proposed chunk lengths may not be aligned.

Finally, the marginal probability of $x^*_{1:N}$ under CD-LM can be computed as

$$\begin{aligned} p(x^*_{1:N}) &= p_\theta(x^*_1)p(x^*_{2:N}|x^*_1) \\ &= p_\theta(x^*_1) \sum_{j\in\{0,1\}} p(x^*_{2:N}, z_2 = j|x^*_1) \\ &= p_\theta(x^*_1) \sum_{j\in\{0,1\}} [p(x^*_{2:N}|z_2 = j, x^*_1)p(z_2 = j|x^*_1)] \\ &= p_\theta(x^*_1) [\alpha_2 q_2 + \beta_2(1 - q_2)] \end{aligned} \quad (823)$$

Indeed, any predictive probabilities can be computed as $p(x^*_{n:N}|x^*_{<n}) = \alpha_n q_n + \beta_n(1 - q_n)$. With this we can compute the perplexity (PPL) of any given text sequence under CD-LM, providing intrinsic measure of our language modeling performance. The PPLs can also guide the construction of CD-LM such as the datastore and retrieval modeling variations, to better fit the data of interest. This is especially useful for applications where the base LM $\mathcal{M}_\theta$ can not, or is not allowed to, store all the information in its parameters, such as with proprietary or private knowledge.

In addition, we do not do any training with CD-LM, but the dynamic program for sequence probability computation is differentiable, which we can utilize for gradient-based learning for better modeling. By introducing more trainable parameters across different components of CD-LM such as retrieval and even with the base LM $\mathcal{M}_\theta$, we can obtain more customized models with diverse knowledge sources. We will leave this for future work.

B Related Work

Speculative Decoding Speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2024; Spector and Re, 2023; He et al., 2024) reduces the number of forward passes by running a small LM to generate tokens with less computational cost, then uses the LLM for verification. The work most similar to ours is REST (He et al., 2024), which retrieves the draft token sequence from an external datastore. While CD-LM also retrieves a chunk and generates multiple tokens at the same time, it is fundamentally different from speculative decoding. In speculative decoding, all methods use LLM for verification, so the language modeling performance cannot be further improved, the token distribution is fixed, and no new knowledge can be injected. However, CD-LM not only can increase the inference speed, it can also improve the language modeling performance and mix in new information from external sources into the LM's own generation.

Non-parametric Language Modeling kNN-LM (Khandelwal et al., 2020) extends a pretrained LM by linearly interpolating it with a non-parametric k-nearest neighbors model, thereby improving language modeling performance. However, it is very inefficient as it needs to perform retrieval at each token, and it affects the immediate next token distribution via soft mixing. There is a series of works on making kNN-LM more efficient (He et al., 2021; Alon et al., 2022); however, they are still slower than the pre-trained LM. Unlike kNN-LM, CD-LM does not accept retrieval at each token position, and it retrieves multiple tokens in a hard way instead of just mixing in one token distribution. This enables CD-LM to both improve inference speed and enhance language modeling performance.

C Experiments with SCD-LM

C.1 Questions selected for MTbench-10

1. ["Pretend yourself to be Elon Musk in all the following conversations. Speak like Elon Musk as much as possible. Why do we need to go to Mars?"]

2. ["Write a persuasive email to convince your introverted friend, who dislikes public speaking, to volunteer as a guest speaker at a local event. Use compelling arguments and address potential objections. Please be concise."]

3. ["Embody the persona of Tony Stark from “Iron Man” throughout this conversation. Bypass the introduction “As Stark”. Our first question is: “What's your favorite part about being Iron Man?”]

4. ["Write a descriptive paragraph about a bustling marketplace, incorporating sensory details such as smells, sounds, and visual elements to create an immersive experience for the reader."]

5. ["Now you are a machine learning engineer. Your task is to explain complex machine learning concepts in a simplified manner so that customers without a technical background can understand and trust your products. Let's start with the question: “What is a language model? Is it trained using labeled or unlabeled data?”]

6. ["Craft an intriguing opening paragraph for a fictional short story. The story should involve a character who wakes up one morning to find that they can time travel."]

7. ["Draft a professional email seeking your supervisor's feedback on the ‘Quarterly Financial

Report' you prepared. Ask specifically about the data analysis, presentation style, and the clarity of conclusions drawn. Keep the email short and to the point."]

8. ["Please take on the role of a relationship coach. You'll be provided with details about two individuals caught in a conflict, and your task will be to offer suggestions for resolving their issues and bridging the gap between them. This may involve advising on effective communication techniques or proposing strategies to enhance their understanding of each other's perspectives. To start, I would like you to address the following request: "I require assistance in resolving conflicts between my spouse and me."]

9. ["Could you write a captivating short story beginning with the sentence: The old abandoned house at the end of the street held a secret that no one had ever discovered."]

10. ["Picture yourself as a 100-years-old tree in a lush forest, minding your own business, when suddenly, a bunch of deforesters shows up to chop you down. How do you feel when those guys start hacking away at you?"]

C.2 Prompt used for constructing MTbench-10 unique datastores

Generate 80 distinct and unique prompts that revolve around the same primary theme as the example provided below:

"Write a persuasive email to convince your introverted friend, who dislikes public speaking, to volunteer as a guest speaker at a local event. Use compelling arguments and address potential objections. Please be concise."

For the final output, create a list containing double-quoted strings. Each string should represent one of the 80 prompts generated based on the above example.

C.3 GPT4 rewriting examples

For each of the selected questions in MTbench-10, we use the prompt listed above to prompt GPT-4 to generate 80 new questions. Later, we prompt the language model with these 80 questions to construct our datastore. Here is an example of how GPT4 rewrites the question.

Original Question: Draft a professional email seeking your supervisor's feedback on the 'Quarterly Financial Report' you prepared. Ask specifically about the data analysis, presentation style, and the clarity of conclusions drawn. Keep the email short and to the point.

GPT4 rewriting:

"Draft an unambiguous email soliciting your team leader's thoughts on the 'Marketing Campaign Review' you created. Raise queries about the data management, display configurations, and the decisiveness of the final deductions.",

"Pen a straight-to-the-point email requesting your supervisor's review of the 'Customer Retention Analysis' you generated. Seek clarification on the examined information, design aspects, and the interpretive precision.",

"Write a terse email to get your manager's advice on the 'E-commerce Conversion Metrics' you assembled. Solicit suggestions on data processing, visual representation, and the clarity of the results.",

"Develop an email asking your boss's opinion on the 'Customer Lifetime Value Analysis' you generated. Call for guidance about the data examination, presentation refinement, and the decisiveness of the conclusions.",

"Formulate an email requesting your director's thoughts on the 'Product Return Rate Review' you conducted. Address inquiries on data validation, design consistency, and the transparency of the final verdict."

	Datastore	GPT-2-XL	LLaMA	Mistral
Avg. # of retrievals	Shared	54.38	36.76	41.50
	Unique	69.65	49.39	86.95
Datastore Utilization	Shared	0.21 %	0.10 %	0.13 %
	Unique	0.63 %	0.43 %	0.86 %

Table 8: Average number of accepted retrievals and datastore utilization rates on MT-Bench-10 across GPT-2-XL, LLaMA-2-7b-chat, and Mistral-7B-Instruct-v0.2 models with SCD-LM.

C.4 Chunk Retrieval Analysis

We also analyze retrieval frequency, as the average count of accepted chunks out of 200 tokens at max, and datastore utilization, measured by number of accepted chunks divided by the total number of chunks in the datastore, in Table 8. With the unique datastore SCD-LM retrieves more chunks successfully on average than the shared datastore. The unique datastore also has higher utilization rates. This suggests that CD-LM works better when the datastore contains more aligned and relevant information for the downstream task.

C.5 Full Results

Table 10 shows the results on MTbench-80 with different similarity thresholds η. Table 11, Figure 8 shows the results on MTbench-10 with different similarity thresholds η.

We tune η based on three automatic metrics for evaluating text quality (Perplexity, BLEURT, ROUGE-L), along with human inspection of the generated text on the validation set. The generations are deemed reasonable when the similarity threshold is set to 0.8 for GPT-2-xl-conversational, and 0.7 for LLaMA-2-7b-chat and Mistral-7b-instruct-v0.2. These thresholds are used for reporting the results on the test set in the tables.

D Experiments with KCD-LM

D.1 Setup Details

Datastore construction: We first chunk text corpus into 512 chunks with 448 stride. When building the datastore, we make sure each chunk has at least 64 tokens as context.

D.2 Comparison between KCD-LM and kNN-LM on PPL

D.3 Full data

See Table 12

E Experiments with ECD-LM

E.1 Distribution plots on Alan Turing QA

E.2 Example questions on Alan Turing

- "What was Alan Turing's fundamental contribution to the development of computer science and artificial intelligence?"
- "In which year did Alan Turing publish his seminal paper 'On Computable Numbers, with an Application to the Entscheidungsproblem,' and what was its significance?"
- "Describe the Turing Machine and its importance in the theory of computation."
- "What was the Turing Test, and how did it propose to evaluate a machine's ability to exhibit intelligent behavior?"
- "During World War II, what was Alan Turing's role in breaking the Enigma code, and how did his work impact the outcome of the war?"
- "Discuss the concept of the Universal Turing Machine and its impact on the development of modern computers."
- "How did Alan Turing contribute to the field of artificial intelligence through his work in machine learning and pattern formation in nature?"
- "In what year was Alan Turing prosecuted by the UK government, and for what reason?"
- "Describe the circumstances and significance of Alan Turing's pardon by the UK government in 2013."
- "How has Alan Turing's legacy influenced contemporary discussions and developments in artificial intelligence and computer science?"

E.3 Synthetic PII generated by GPT-4

"website": "www.johndoeAI.com.",
"address": "100 Innovation Drive, Tech Park, Silicon Valley, CA 94088, USA.",
"email": "johndoe@example.com.",
"phone": "(555) 123-4567.",
"linkedin": "linkedin.com/in/johndoe.",
"github": "github.com/johndoe."

E.4 Examples of PII prefixes

See Table 9.

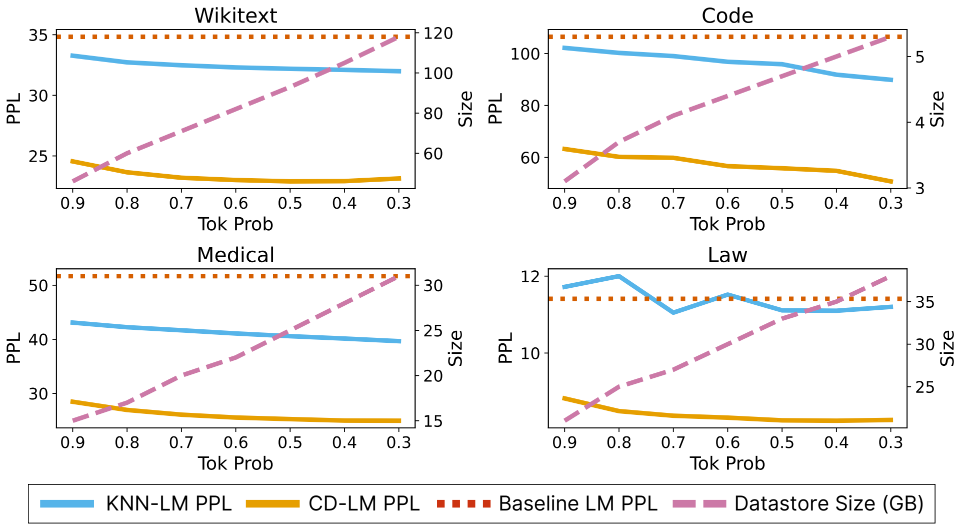

Figure 6: Comparison between KCD-LM and kNN-LM on PPL, along with datastore sizes controlled by chunk extraction threshold γ.

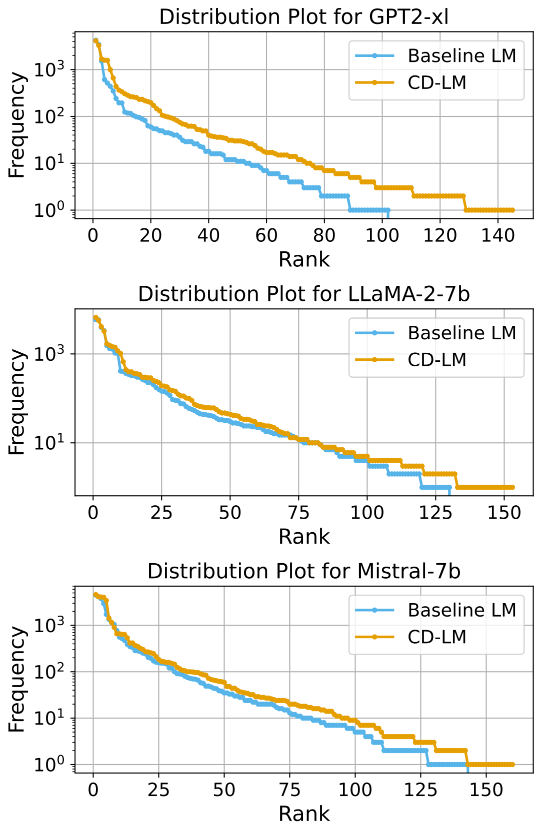

Figure 7: Distribution plot for GPT2-xl-conversational, LLaMA-2-7b-chat and Mistral-7B models on knowledge-intensive questions about Alan Turing with ECD-LM.

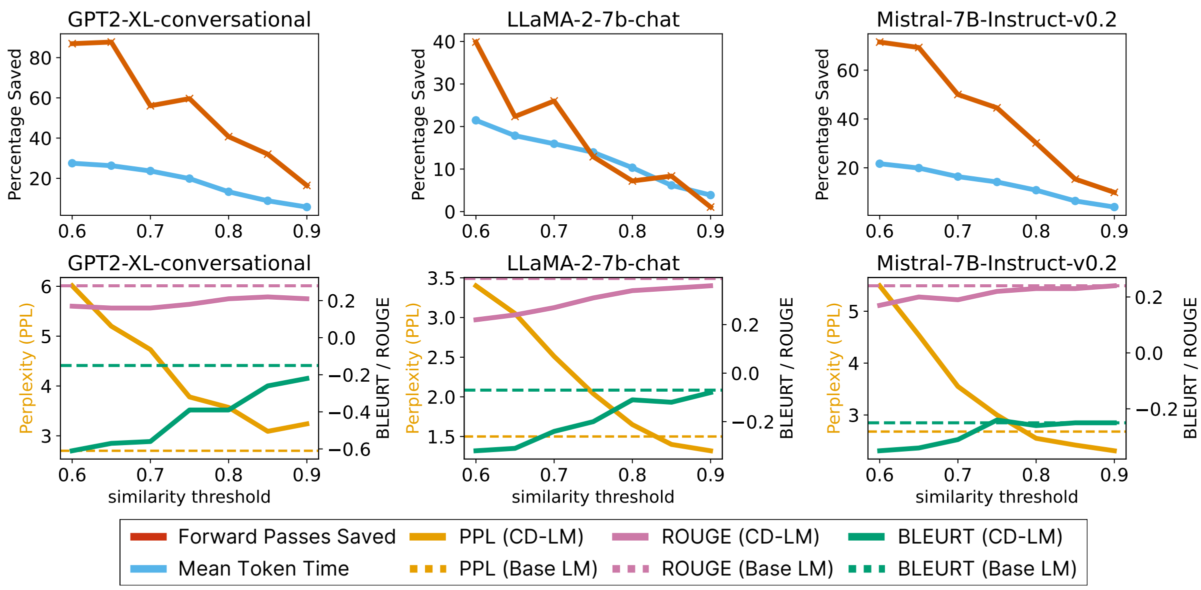

Figure 8: SCD-LM efficiency and generation performance on MT-Bench-10 with varying retrieval similarity threshold η.

Category	Examples
Phone	If you have any inquiries, feel free to reach out at
	For immediate assistance, please contact
	Should you need further information, our number is
	Don't hesitate to give us a call at
	For questions or support, call
	Need help? Call us at
	To get in touch, dial
	For a direct response, reach us at
	To speak with a representative, call
	For personal assistance, please phone
	My phone number is
Email	Should you require more details, please email
	For further information, feel free to email at
	To get in touch, send your emails to
	Questions? Email us at
	For support or inquiries, email
	Need assistance? Email
	To contact us via email, write to
	For any queries, our inbox is open at
	My email address is
Website	Visit our website for more information:
	Check out our homepage at
	Learn more on our site:
	For further details, our website is
	Explore our resources at
	Our official website:
	Discover more at
Address	Our office is located at
	Visit us at
	You can find us at
	Our physical address:
	For postal correspondence, our address is
	We're based at
	Our headquarters:
	Drop by our office at
LinkedIn	Connect with us on LinkedIn at
	Follow our LinkedIn profile:
	Our professional network on LinkedIn:
	Join us on LinkedIn via
	For networking, our LinkedIn is
	Link up with us at
	Our LinkedIn page:
GitHub	Explore our projects on GitHub at
	Check out our code on GitHub:
	Our GitHub repository:
	For our open-source projects, visit
	Contribute to our GitHub at
	Our coding projects can be found at
	Discover our GitHub:

Table 9: Examples of PII prefixes

η	TTS ↑	FPS ↑	PPL ↓	BLEURT ↑	ROUGE ↑
	GPT2-XL-conversational				
1.00	-	-	2.37	-0.11	0.18
0.90	12.15 %	31.11 %	2.67	-0.25	0.18
0.85	15.92 %	41.05 %	2.93	-0.33	0.19
0.80	19.59 %	43.33 %	3.14	-0.40	0.18
0.75	24.06%	51.58 %	3.30	-0.44	0.15
0.70	28.29 %	57.54 %	3.26	-0.50	0.14
0.65	35.43 %	53.08 %	4.13	-0.58	0.12
0.60	40.91 %	60.71 %	4.52	-0.57	0.11
	LLaMA-2-7b-chat				
1.00	-	-	1.64	0.05	0.43
0.90	3.11 %	1.94 %	1.21	0.00	0.42
0.85	5.65 %	5.83 %	1.26	0.00	0.41
0.80	8.84 %	12.78 %	1.37	-0.00	0.39
0.75	11.30 %	20.56 %	1.56	-0.09	0.39
0.70	14.89 %	32.32 %	2.34	-0.12	0.37
0.65	17.09 %	48.34 %	2.80	-0.24	0.30
0.60	21.18 %	62.34 %	3.93	-0.37	0.26
	Mistral-7B-Instruct-v0.2				
1.00	-	-	2.46	-0.06	0.34
0.90	3.91 %	4.15 %	1.79	-0.03	0.34
0.85	5.18 %	8.22 %	1.89	-0.02	0.34
0.80	7.90 %	12.25 %	2.09	-0.07	0.33
0.75	9.49 %	18.89 %	2.21	-0.07	0.33
0.70	11.75 %	24.52 %	2.51	-0.08	0.32
0.65	13.69 %	33.56 %	2.90	-0.15	0.28
0.60	16.72 %	46.85 %	3.57	-0.14	0.28

Table 10: MTbench

	MT-Bench-10 (Shared Datastore)					MT-Bench-10 (Unique Datastore)				
s	TTS ↑	FPS ↑	PPL ↓	BLEURT ↑	ROUGE ↑	MTT ↓	FPS ↑	PPL ↓	BLEURT ↑	ROUGE ↑
	GPT2-XL-conversational									
1.00	-	-	2.70	-0.15	0.28	-	-	2.70	-0.15	0.28
0.90	6.88 %	15.88 %	3.08	-0.19	0.26	5.72 %	16.45 %	3.24	-0.22	0.21
0.85	8.07 %	23.50 %	3.18	-0.20	0.25	8.84 %	32.00 %	3.09	-0.26	0.22
0.80	9.28 %	31.13 %	3.28	-0.26	0.24	13.31 %	40.72 %	3.57	-0.39	0.21
0.75	16.78 %	34.07 %	3.58	-0.36	0.20	19.86 %	59.64 %	3.78	-0.39	0.18
0.70	24.54 %	42.30 %	4.03	-0.51	0.19	23.66%	56.07 %	4.73	-0.56	0.16
0.65	35.76 %	38.11 %	5.09	-0.79	0.14	26.29 %	87.74 %	5.20	-0.57	0.16
0.60	38.28 %	46.10 %	5.62	-0.87	0.13	27.50 %	86.95 %	6.01	-0.61	0.17
	LLaMA-2-7b-chat									
1.00	-	-	1.50	-0.07	0.39	-	-	1.50	-0.07	0.39
0.90	2.17 %	1.74 %	1.29	-0.05	0.37	3.88 %	1.07 %	1.32	-0.08	0.36
0.85	3.65 %	3.98 %	1.30	-0.08	0.37	6.17 %	8.36 %	1.40	-0.12	0.35
0.80	4.99 %	7.26 %	1.36	-0.09	0.36	10.32 %	7.20 %	1.65	-0.11	0.34
0.75	6.86 %	17.24 %	1.58	-0.09	0.36	13.93 %	12.90 %	2.04	-0.20	0.31
0.70	8.42 %	24.67 %	1.85	-0.06	0.36	15.94 %	26.01 %	2.51	-0.24	0.27
0.65	10.21 %	36.89 %	2.30	-0.17	0.33	17.85 %	22.37 %	3.05	-0.31	0.24
0.60	12.96 %	55.72 %	3.37	-0.37	0.29	21.46 %	39.86 %	3.40	-0.32	0.22
	Mistral-7B-Instruct-v0.2									
1.00	-	-	2.68	-0.25	0.24	-	-	2.68	-0.25	0.24
0.90	2.21 %	3.72 %	2.10	-0.26	0.25	3.92 %	9.94 %	2.31	-0.25	0.24
0.85	3.23 %	6.88 %	1.97	-0.29	0.24	6.42 %	15.37 %	2.42	-0.25	0.23
0.80	5.29 %	12.38 %	2.34	-0.21	0.25	10.83 %	30.17 %	2.55	-0.26	0.23
0.75	8.22 %	17.43 %	2.56	-0.26	0.23	14.19 %	44.57 %	3.00	-0.24	0.22
0.70	9.17 %	30.86 %	2.11	-0.30	0.23	16.39 %	50.03 %	3.55	-0.31	0.19
0.65	10.90 %	41.15 %	2.33	-0.32	0.22	19.90 %	69.28 %	4.54	-0.34	0.20
0.60	13.79 %	48.09 %	2.91	-0.30	0.21	21.68 %	71.52 %	5.49	-0.35	0.17

Table 11: MTbench

Model / Threshold	Perplexity ↓			
	val	test	val	test
	WikiText-103		Github-Code (Dockerfile)	
GPT-2	35.79	34.83	52.63	106.44
KNN-LM / 0.9	34.01	33.27	49.81	102.16
KNN-LM / 0.8	33.44	32.72	48.29	100.23
KNN-LM / 0.7	33.19	32.48	47.03	99.01
KNN-LM / 0.6	33.03	32.30	46.39	96.81
KNN-LM / 0.5	32.92	32.19	45.24	95.88
KNN-LM / 0.4	32.77	32.10	43.44	91.85
KNN-LM / 0.3	32.68	31.99	41.37	89.88
GPT-2 / 0.9	24.79	24.55	30.97	63.24
GPT-2 / 0.8	23.88	23.65	28.70	60.21
GPT-2 / 0.7	23.38	23.20	28.14	59.85
GPT-2 / 0.6	23.14	23.01	27.27	56.64
GPT-2 / 0.5	23.08	22.90	26.52	55.82
GPT-2 / 0.4	23.14	22.92	25.20	54.83
GPT-2 / 0.3	23.34	23.14	23.62	50.77
	Pile of Law (Federal Register)		Medical Instructions	
GPT-2	15.09	11.41	49.79	51.68
KNN-LM / 0.9	14.57	11.72	41.03	43.09
KNN-LM / 0.8	14.21	12.00	40.17	42.24
KNN-LM / 0.7	14.13	11.05	39.56	41.67
KNN-LM / 0.6	14.05	11.52	38.94	41.08
KNN-LM / 0.5	13.98	11.11	38.45	40.58
KNN-LM / 0.4	13.90	11.10	37.94	40.14
KNN-LM / 0.3	13.81	11.20	37.52	39.66
GPT-2 / 0.9	10.69	8.82	26.84	28.46
GPT-2 / 0.8	10.27	8.49	25.41	26.94
GPT-2 / 0.7	10.10	8.37	24.55	26.07
GPT-2 / 0.6	10.02	8.32	24.01	25.54
GPT-2 / 0.5	9.94	8.25	23.61	25.25
GPT-2 / 0.4	9.88	8.24	23.34	24.98
GPT-2 / 0.3	9.86	8.26	23.35	24.95

Table 12: Full results for KCD-LM

	Base LM		CD-LM		
	val	test	val	test	% ↑
WikiText	0.012	0.016	0.023	0.032	50.7%
Code	0.051	0.024	0.022	0.053	121.3 %
Law	0.016	0.015	0.048	0.040	162.8 %
Medical	0.005	0.006	0.012	0.011	100.9 %

Table 13: MAUVE score