Natural Language Processing (2025), 1–35
doi:10.1017/nlp.2025.10010

CAMBRIDGE UNIVERSITY PRESS

ARTICLE

How to quickly select good in-context examples in large language models for data-to-text tasks?

Yulong Li[1], Jiaoyun Yang[1], Lili Jiang[2], Shuo Liu[1] and Ning An[1]

[1]Hefei University of Technology, Hefei, China and [2]Deparment of Computing Science, Umeå University, Umeå, Sweden
Corresponding author: Jiaoyun Yang; Email: jiaoyun@hfut.edu.cn

(Received 11 August 2024; revised 28 June 2025; accepted 13 September 2025)

Abstract

In the realm of data-to-text generation tasks, the use of large language models (LLMs) has become common practice, yielding fluent and coherent outputs. Existing literature highlights that the quality of in-context examples significantly influences the empirical performance of these models, making the efficient selection of high-quality examples crucial. We hypothesize that the quality of these examples is primarily determined by two properties: their similarity to the input data and their diversity from one another. Based on this insight, we introduce a novel approach, Double Clustering-based In-Context Example Selection, specifically designed for data-to-text generation tasks. Our method involves two distinct clustering stages. The first stage aims to maximize the similarity between the in-context examples and the input data. The second stage ensures diversity among the selected in-context examples. Additionally, we have developed a batched generation method to enhance the token usage efficiency of LLMs. Experimental results demonstrate that, compared to traditional methods of selecting in-context learning samples, our approach significantly improves both time efficiency and token utilization while maintaining accuracy.

Keywords: in-context learning; data-to-text; large language models; double clustering; batched generation

1. Introduction

Data-to-text generation, the process of converting structured data into readable human language, is a rapidly growing research area with significant applications in domains such as automated report generation and customer service (Gatt and Krahmer 2018). This complex transformation involves translating various data sources such as record databases, spreadsheets, and knowledge bases into natural language outputs (Gardent *et al.* 2017). For example, given two triples `(Aarhus_Airport, cityServed, "Aarhus, Denmark")` and `(Aarhus_Airport, runwayLength, 2,777 m)`, a data-to-text system should produce a sentence such as "Aarhus Airport, which serves the city of Aarhus in Denmark, has a runway that is 2,777 m long."

The challenges in data-to-text generation include sophisticated context understanding, diverse data structure identification, and complex sentence construction. Developing a data-to-text generation model often requires training or fine-tuning with a substantial amount of data. Moreover, this process is further complicated by the need to train separate models for different input data formats and output sentence styles. Large Language Models (LLMs) (Brown *et al.* 2020; Raffel *et al.* 2020) offer a solution to these challenges by effectively understanding both natural language and structured text, as well as generating coherent and fluent outputs (OpenAI 2023; Zhao *et al.* 2023c).

While some heuristic methods have been used to create prompts for data-to-text generation (Zhao *et al.* 2023c), these methods can be time-consuming and labor-intensive. Therefore,

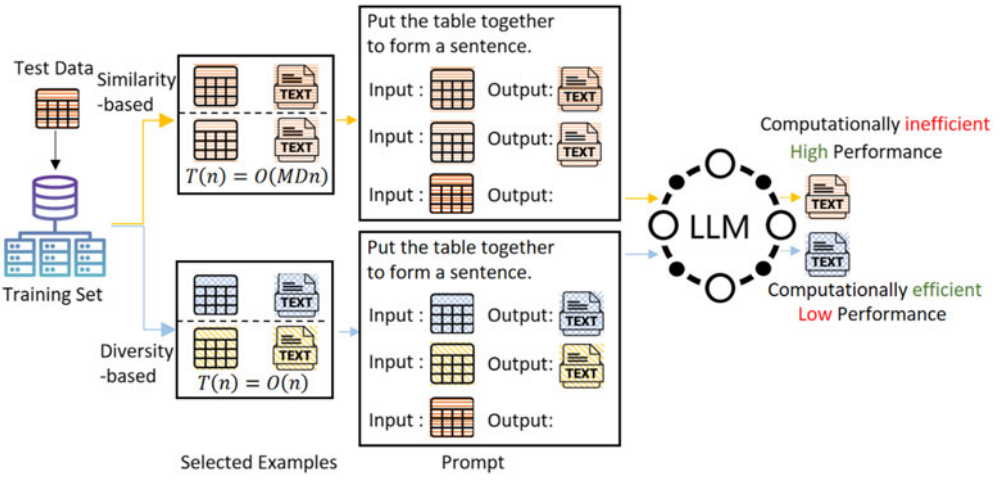

Figure 1. Flowchart of different in-context learning methods for data-to-text. Different colored icons represent the semantic representations of data/text. The similarity-based method chooses demonstrations that are more similar to the input data and achieves better output results, but it has higher time complexity (*M* is the size of the training set, *n* is the size of the test set, *D* is the size of the representation vector). In contrast, the diversity-based method has lower time complexity but produces poorer output results.

automatic prompt generation is needed to guide LLMs in meeting specific text requirements. In-Context Learning (ICL) (Brown *et al.* 2020; Dong *et al.* 2024) provides a simple yet effective approach by incorporating input–output demonstrations into the prompt, thereby enhancing generative performance.

Recent studies have shown that the selection, format, and order of in-context examples in the prompt can significantly affect the model's output (Zhao *et al.* 2021; Min *et al.* 2022; Liu *et al.* 2022b). Figure 1 illustrates how different choices of in-context examples influence the generation results of LLM.

Consequently, methods for prompt construction based on in-context example selection have emerged. These methods can be broadly categorized into unsupervised methods (Lu *et al.* 2022; Su *et al.* 2022; Liu *et al.* 2022b) and supervised methods (Rubin, Herzig, and Berant, 2022). Supervised methods require fine-tuning the model for different tasks, which is not only costly but also impractical, especially when the model is accessed only via API calls. Unsupervised methods can be further divided into similarity-based (Liu *et al.* 2022b) and diversity-based approaches (Zhang *et al.* 2022).

Similarity-based methods aim to select training samples most similar to the test input for use as in-context examples. This approach often yields favorable results in many cases. However, it requires comparing the test input with all candidate samples in the training set. As a result, it incurs high computational costs for large datasets.

In contrast, diversity-based methods aim to include as many diverse examples as possible to cover a broader range of potential input distributions. These methods typically cluster samples and select representative examples from each cluster, or design and apply heuristic strategies to choose diverse examples. Compared to similarity-based methods, diversity-based methods use the same set of in-context examples for different test inputs, which in turn improves computational efficiency. However, since the examples in the prompt are not closely related to the test input, the generated text may lack precision.

The generation results in data-to-text tasks can be significantly influenced by the structured data and reference texts in the in-context examples. If the characteristics of the in-context examples differ significantly from those of the test input, the generation results may degrade, particularly in smaller models or for tasks requiring high precision. Improving the efficiency of

utilizing LLMs for data-to-text generation remains an important area of research, particularly for large-scale datasets, where balancing efficiency with the accuracy and consistency of generated results presents unique challenges. To address these challenges, we propose an efficient prompt generation method through in-context example selection: Double Clustering-based in-Context example Selection (DCCS), tailored for data-to-text tasks. The contributions of our research are as follows:

a) We propose a two-step in-context example selection method based on double clustering. In the preprocessing phase, we perform two clustering operations on the training dataset. The first clustering process ensures data similarity among the in-context examples, while the second clustering process ensures the diversity of the reference texts. In the inference phase, we simplify the in-context example selection process to a cluster selection process for efficiency.
b) We propose a batched generation method based on the DCCS approach, which groups similar samples from the first clustering into batched prompts based on the batch size and then utilizes a LLM for generation. This method effectively improves token utilization while ensuring the accuracy of the generated results.
c) We conducted experiments on four data-to-text datasets. The experimental results indicate that prompts constructed using our in-context example selection method achieved high accuracy across multiple LLMs. Additionally, our method significantly reduces the time and tokens needed for prompt generation in LLMs.

The subsequent sections provide a comprehensive discourse on our proposed methodology, the experimental setup, and the consequent findings. These sections elucidate the capacity of our DCCS approach to advance data-to-text generation.

2. Related work

2.1 Data-to-text generation

Data-to-text generation focuses on converting structured data into coherent textual descriptions. This task has garnered substantial attention due to its applications in sports commentary, biographical text generation, and open-domain table summarization. Commonly used datasets include WebNLG (Gardent *et al.* 2017) and DART (Nan *et al.* 2021), with additional studies addressing related challenges (Wen *et al.* 2015; Lebret, Grangier, and Auli 2016; Wiseman, Shieber, and Rush 2017). In addition to open-domain table summarization, converting tables into text and learning embeddings of tables have been shown to benefit open-domain question answering and information retrieval, for example, by enabling dense retrieval that embeds questions and tables in a shared vector space (Deng, Zhang, and Balog 2019; Herzig *et al.* 2021).

Initial research on data-to-text generation predominantly involved template filling and rule-based approaches (Hallett, Power, and Scott 2006; Turner *et al.* 2008). While effective for narrowly defined, domain-specific tasks, these methods often struggled to produce diverse and natural outputs, limiting their broader applicability. Subsequent work shifted toward fine-tuning models such as T5 (Raffel *et al.* 2020) and GPT-2 (Radford *et al.* 2019) on task-specific datasets (Li and Liang 2021; Clive, Cao, and Rei 2022). For instance, LOFT (Zhao *et al.* 2023a) employs logic forms as fact verifiers and content planners to enhance controllability, faithfulness, and diversity. PLOG (Liu *et al.* 2022a) continuously pre-trains text generation models on a data-to-logic-form generation task to improve output fidelity. Despite their success, these methods typically require separate training for each new task. This approach is time-consuming, costly, and difficult to scale, especially in rapidly evolving fields where task requirements frequently change.

More recent research leverages the inherent capabilities of LLMS, such as employing chain-of-thought prompts (Zhao *et al.* 2023b), which guide LLMs through step-by-step reasoning to

enhance faithfulness and coherence in data-to-text tasks. However, such methods require complex prompt engineering and substantial prior knowledge, limiting their efficiency.

To address these limitations, an emerging paradigm is In-Context Learning (ICL). ICL enables LLMs to adapt to new tasks by including a few task-specific examples in the prompt, eliminating the need for fine-tuning. This approach offers a flexible and scalable solution for data-to-text generation and serves as the basis for our investigation.

2.2 In-context learning

With LLMs widely deployed across various tasks, their ability to learn from a few examples in the prompt has garnered substantial attention. ICL enhances the efficacy of these models without extensive fine-tuning (Brown *et al.* 2020; Wei *et al.* 2022), enabling LLMs to perform diverse tasks, such as web browsing and coding (Nakano *et al.* 2021; Chen *et al.* 2021). The OpenICL framework provides a unified platform that simplifies ICL implementation (Wu *et al.* 2023b).

The effectiveness of ICL heavily relies on the selection and formatting of demonstrations (Zhao *et al.* 2021; Min *et al.* 2022; Liu *et al.* 2022b), which has prompted the development of optimization strategies. These strategies can be broadly categorized into supervised and unsupervised methods (Dong *et al.* 2024). Supervised methods train a scorer to select in-context examples using supervision signals (Rubin *et al.* 2022; Li *et al.* 2023), leveraging feedback from language model inference to adaptively identify suitable demonstrations. CEIL (Ye *et al.* 2023) and MoD (Wang *et al.* 2024) model the selection of the exemplar set and train one or multiple retrievers to score the exemplar set. While these methods often yield better performance, they are challenging to implement for black-box LLMs accessed via token-level APIs and can be computationally expensive.

In contrast, unsupervised methods rely on heuristics such as similarity-based, diversity-based, or entropy-based approaches (Lu *et al.* 2022; Liu *et al.* 2022b; Levy, Bogin, and Berant 2023) for demonstration selection. A straightforward approach is to choose the nearest neighbors of input instances based on their similarities, using metrics such as L2 distance or cosine similarity computed from sentence embeddings (Liu *et al.* 2022b; Tanwar *et al.* 2023; Qin *et al.* 2024). Beyond distance metrics, other unsupervised approaches include mutual information (Sorensen *et al.* 2022) and perplexity (Gonen *et al.* 2023), which have shown promise for prompt selection without labeled data or task-specific LLMs. However, most of these methods are designed for general question-answering tasks and do not consider, in data-to-text generation, the differing impacts that in-context examples' structured data and reference text components have on the model's output quality.

In this work, we propose a novel unsupervised demonstration selection strategy that reduces reliance on heuristics while improving efficiency and reducing token overhead. Our approach retains the simplicity and broad applicability of unsupervised methods while achieving better performance for data-to-text tasks.

3. Methodology

In this section, we present our Double Clustering-based in-Context Example Selection (DCCS) methodology, aimed at rapidly selecting high-quality in-context examples to improve the efficiency of data-to-text tasks. Building on DCCS, we further propose the DCCS-Batch method, which enhances token utilization.

3.1 Definitions and problem formulation

Let the training set consist of M samples denoted by $S = (X_{\text{train}}, Y_{\text{train}})$, where $X_{\text{train}} = \{\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_M\}$ contains the *data* portions of the samples, and $Y_{\text{train}} = \{\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_M\}$ contains the corresponding reference *texts*. Given a test input x_{test}, our goal is to produce an output

$y_{\text{test}} = \mathscr{M}(C \oplus x_{\text{test}})$, where $\mathscr{M}$ denotes a large language model (e.g., GPT-3.5) and C is composed of an instruction I along with m in-context examples, denoted by $s(x_i, y_i)$. Formally,

$$C = \{I, \underbrace{s(x_1, y_1), s(x_2, y_2), \ldots, s(x_m, y_m)}_{\text{context}}\}. \tag{1}$$

Conventional unsupervised in-context example selectors usually optimize a single dimension – either semantic similarity (e.g., KATE Liu *et al.* 2022b) or diversity (e.g., choosing cluster centroids Zhang *et al.* 2022). In data-to-text generation, each approach exhibits a characteristic weakness. Diversity-only schemes often return demonstrations that are too remote from the test instance, yielding sub-optimal outputs. Similarity-only schemes, by contrast, require evaluating the similarity between the test instance and every training sample, then taking the top-k neighbors; although such exhaustive search can sometimes achieve the best generations, its inference-time complexity is prohibitive when the training set is large. To overcome these issues, we compare the test instance with only a small subset of the training data while still recovering near-optimal demonstrations. Our key insights are as follows: (i) examples whose data fields are sufficiently similar to the test instance are beneficial; (ii) the absolutely closest neighbors are unnecessary – near-enough examples suffice; and (iii) maximizing diversity in the text fields helps the model cover a wide range of reference-level linguistic patterns.

3.2 Method overview

Our approach leverages both *data-level similarity* and *text-level diversity* to optimize the selection process:

Data-Level Similarity: We cluster the training samples into K groups based on the semantic representations of their data components, and Section 3.3.2 details how K is selected. Samples within the same cluster are considered similar in terms of their data characteristics, while those from different clusters are treated as dissimilar. For a test input, we compare its data representation only with the K cluster centers, thereby significantly reducing the computational cost of similarity comparisons compared to evaluating all samples in the training set.

Text-Level Diversity: We argue that if two in-context examples are highly similar in their textual content, including only one of them is sufficient to represent that semantic segment. To ensure textual diversity, we perform a second clustering step within each relevant data cluster; this time focusing on the text components of the samples. The center of each text cluster is then selected as an in-context example, enabling broader coverage of possible expressions while keeping the example quantity unchanged.

Conventional diversity-driven selectors – for example, Determinantal Point Process (DPP) (Wu *et al.* 2023a) and centroid-based clustering (Zhang *et al.* 2022; Levy *et al.* 2023) – pursue diversity alone and neglect semantic alignment, so demonstrations that are overly dissimilar to the test instance can undermine data-to-text generation. DCCS mitigates this shortcoming with a two-stage clustering scheme: the first stage filters out irrelevant samples by data similarity, and the second stage enforces textual diversity within the retained subset. This dual focus simultaneously preserves relevance and expands linguistic coverage, yielding higher-quality outputs. The proposed method also offers substantial efficiency gains. Instead of comparing a test instance with all M training samples, DCCS consults only the K data-cluster centroids ($K \ll M$) and supports *Batched Generation*, whereby a single demonstration set is reused for every test input in the same cluster. This sharply lowers prompt-construction time and token consumption – an important benefit for LLM APIs billed per token. Unlike supervised selectors such as CEIL (Ye *et al.* 2023) and MoD (Wang *et al.* 2024), DCCS requires no additional training and thus operates directly on black-box APIs.

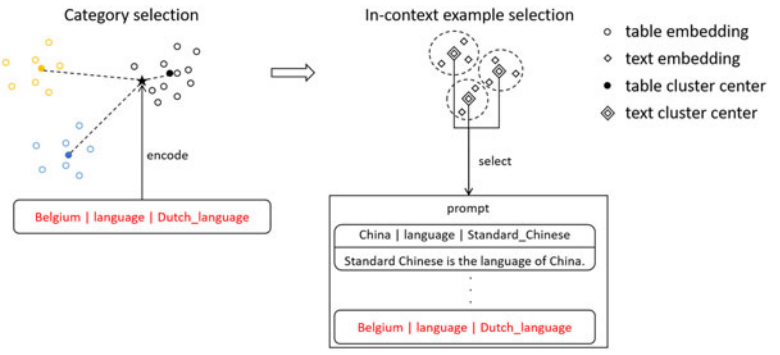

Figure 2. The schematic representation of DCCS. The training set is initially stratified into K primary clusters based on the embeddings of the data. Each primary cluster subsequently undergoes sub-clustering into m categories based on the embeddings of the reference texts. The m centroid samples are selected as candidate in-context examples for each primary cluster. During the inference phase, the test data x is encoded and assessed for similarity against the K cluster centers, and the candidate in-context examples from the proximal category is selected.

3.3 Double clustering based in-context example selection

For data-to-text tasks, our intention is to ensure that the context in the prompt is as pertinent to the data of x_{test} as possible, while also possessing maximum diversity in reference texts. Thus, we stratify the in-context example selection task into two steps: (i) Cluster selection for input data, (ii) In-context example selection for reference text. In the cluster selection step, we select a subset of samples whose data content is similar to the given input semantics. In the in-context example selection step, we select training samples with diverse descriptions from the subset to construct a prompt.

This procedure is implemented through two separate clustering operations. The first clustering operation divides the original training set into different categories. The second clustering operation identifies the most diverse in-context examples. We opted for the K-means algorithm for both clustering operations due to its simplicity, swift convergence, and straightforward implementation (Yuan and Yang 2019). The DCCS method is visualized in Figure 2. We divided the whole generation process into two phases: the preprocessing phase and the inference phase, as shown in Algorithms 1 and 2.

The preprocessing phase involves clustering the training set into K primary clusters, each of which is further subdivided into m secondary clusters through a subsequent round of clustering. After the double clustering, the centroids of the first clusters are recorded for future comparison with input samples, while the m samples proximate to the centroids of the m secondary clusters are retained, forming the candidate in-context example sets.

The inference phase involves calculating the distances between the input samples and the centroids of the K clusters. The candidate in-context example sets from the closest cluster are then selected to generate the prompt.

3.3.1 Preprocessing phase

We utilize a smaller Pre-trained Language Model (PLM) to encode X_{train} in the embedding space, resulting $X_{emb} \in R^{M \times D}$, where M represents the number of sentences and D is the embedding dimension. For this study, we employed RoBERTa-large model (Liu *et al.* 2019) as the PLM, using the pooler output for sentence embedding, with $D = 1024$.

$$X_{emb} = encode(X_{train}) \tag{2}$$

By using the PLM, we obtain the encoded representation X_{emb}, which captures the semantic information of training set.

Algorithm 1: Preprocess of DCCS

Input: Training set $S = \{(x_i, y_i)\}_{i=1}^{M}$; Encoder ENC;
Max clusters $K_{\max}$; In-context examples per prompt m
Output: Data-cluster centers $\{\mu_k\}$; Candidate In-context example sets $\{C_k\}$

Encode data part: $X_{emb} \leftarrow \texttt{ENC}(\{x_i\})$
for $K = 2$ **to** $K_{\max}$ **do**
    $\{\mu_k^{(K)}\} \leftarrow \texttt{KMeans}(X_{emb}, K)$;
    Compute silhouette score $Silhouette_K$;
end
$K^\star \leftarrow \arg\max_K Silhouette_K$ `// number of first clusters`
$\{\mu_k\} \leftarrow \texttt{KMeans}(X_{emb}, K^\star)$
for $k = 1$ **to** $K^\star$ **do**
    $Y^k \leftarrow \{(x_i, y_i) \mid x_i \rightarrow \mu_k\}$
    Encode reference text part: $\mathbf{T} \leftarrow \texttt{ENC}(\{y \mid (x, y) \in Y^k\})$
    $\{c_j^k\}_{j=1}^m \leftarrow \texttt{KMeans}(\mathbf{T}, m)$
    $C_k \leftarrow m$ nearest pairs to $\{c_j^k\}$ `// cache In-context example sets`
end
return $\{\mu_k\}, \{C_k\}$

Algorithm 2: Inference of DCCS

Input: Test input x_{test}; Encoder ENC; Candidate In-context example sets $\{C_k\}$

Output: Generated text y_{test}

$x_{\text{emb}} \leftarrow \texttt{ENC}(x_{\text{test}})$
$k^\star \leftarrow \arg\min_k d(x_{\text{emb}}, \mu_k)$ $C \leftarrow C_{k^\star}$
prompt $\leftarrow C \oplus x_{\text{test}}$
$y_{test} \leftarrow \mathscr{M}$(prompt)
return y_{test}

In the first clustering step, X_{emb} is partitioned into K categories, yielding K cluster centers. The selection of K is based on the comparison of silhouette coefficients (Kaufman and Rousseeuw 2009), considering different numbers of clusters. We select the number of clusters that corresponds to the maximum silhouette coefficient value as optimal. The clustering procedure terminates once the relative decrease in inertia falls below 10^{-4} or after 300 Lloyd iterations, whichever occurs first.

$$\mu_1, \ldots, \mu_K = \arg \min_{\mu_1, \ldots, \mu_K} \sum_{x_i \in X_{emb}} \min_j \left(dis(x_i, \mu_j) \right) \tag{3}$$

Here, $dis()$ denotes the euclidean distance. Based on the clustering results of X_{train}, Y_{train} is also partitioned into K categories.

$$Y^k = \{y_i | dis(y_i, \mu_k) < dis(y_i, \mu_j), j \neq k\} \tag{4}$$

Algorithm 3: Inference of DCCS-Batch

Input: Test set $X_{test} = \{x^{(1)}, \ldots, x^{(N)}\}$;Encoder ENC;
Candidate In-context example sets $\{C_k\}$; Data-cluster centers $\{\mu_k\}$; batch size limit n
Output: Generated texts $\{y^{(t)}\}_{t=1}^{N}$

foreach $x^{(t)} \in X_{test}$ do
  $x_{emb}^{(t)} \leftarrow \texttt{ENC}(x^{(t)})$;
  $k^{(t)} \leftarrow \arg\min_k d(x_{emb}^{(t)}, \mu_k)$;
  $\mathscr{B}_{k^{(t)}}$.append$(x^{(t)})$;
end
foreach $k \in \{1, \ldots, K\}$ do
  foreach *batch* $B \subseteq \mathscr{B}_k$ ***with*** $|B| \leq n$ do
    $C \leftarrow C_k$ prompt $\leftarrow C \,\|\, \text{concat}(B)$;
    $\{y^{(t)}\}_{x^{(t)} \in B} \leftarrow \mathscr{M}(\text{prompt})$;
  end
end
return $\{y^{(t)}\}_{t=1}^{N}$

The same PLM used in the previous step encodes Y^k in the embedding space.

$$Y_{emb}^k = encode(Y^k) \tag{5}$$

In the second clustering step, we select m samples that can encompass all samples in each category with minimal distance. Based on preliminary experiments, we observed that increasing the number of in-context examples beyond ten rarely leads to additional performance improvements. Accordingly, we conduct experiments under both 5-shot and 10-shot settings, with the 10-shot results reported in the appendix.

$$y_1^k, \ldots, y_m^k = \arg \min_{y_1^k, \ldots, y_m^k} \sum_{y_i^k \in Y_{emb}^k} \min_j \left(dis(y_i^k, y_j^k) \right) \tag{6}$$

For each category, the candidate in-context example set, denoted as C_k, is constructed. These in-context example sets are indexed by cluster number and can, therefore, be accessed quickly using the corresponding category index.

$$C_k = \{I, s(x_1^k, y_1^k), s(x_2^k, y_2^k), \ldots, s(x_m^k, y_m^k)\} \tag{7}$$

3.3.2 Determining the number of clusters

To determine the number of clusters in the first clustering step, we use the Silhouette Coefficient Rousseeuw (1987), a metric that evaluates the effectiveness of a clustering algorithm by combining measures of cohesion and separation. This helps identify the optimal number of clusters (K) for k-means clustering. The coefficient ranges from -1 to 1: values close to 1 suggest that a sample is well matched to its own cluster and poorly matched to neighboring clusters; values near 0 indicate proximity to the boundary between clusters; and negative values imply potential misclassification.

For each sample i, two quantities are computed: $a(i)$, the mean distance between i and all other points in the same cluster (measuring cohesion); and $b(i)$, the mean distance between i and all points in the nearest different cluster (measuring separation). The Silhouette Coefficient $s(i)$ for each sample is calculated as follows:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

The average Silhouette Coefficient across all samples serves as an evaluation metric for selecting the value of K that maximizes clustering quality. This method provides an objective and efficient means of determining the appropriate number of clusters for k-means clustering.

3.3.3 Inference phase

When generating text for the input data x, we propose the following method:
Encoding. We utilize PLM to encode the information present in the input data x.

$$x_{emb} = encode(x) \tag{8}$$

Selecting cluster. We compare the encoded representation x_{emb} with the clustering centers obtained from the training set. This comparison allows us to determine the closest cluster to which the input data belongs. The index k is selected as the cluster that minimizes the distance between x_{emb} and the cluster center μ_k. By assigning the input data to the appropriate cluster, we can effectively categorize the test data into different clusters.

$$k = \arg\min_k (x_{emb} - \mu_k) \tag{9}$$

Concatenating prompt. After identifying the domain-specific cluster C_k, we select it as the representative set of in-context examples. To construct a prompt for text generation, we concatenate C_k with the original input data x. This prompt serves as a comprehensive input to guide the LLM in generating coherent and relevant text outputs. After obtaining these in-context examples, we concatenate them into a complete prompt following the format illustrated in Figure 4 and Table 1.

Generating output. The constructed prompt, composed of the cluster examples C_k and the input data x, is fed into LLM for text generation. The LLM utilizes the contextual information provided by the prompt to generate the desired output y.

$$y = \mathscr{M}(C_k \oplus x) \tag{10}$$

3.4 DCCS-batch

Traditional KNN-based in-context learning example selection methods require a separate selection process for each test case, as the closest k samples in the training set vary for each test sample. However, the DCCS method allows for the same context samples to be used for different test samples if they are classified into the same cluster. This enables us to combine test samples with identical in-context examples into a single prompt, allowing the LLM to perform text generation for multiple data points simultaneously. Figure 3 illustrates the differences between our batched generation method and the traditional single-generation method. This batched generation method significantly improves the token utilization efficiency of LLMs. Figures 4 and 5, respectively, present examples of prompts for the single-generation and batched-generation methods, with each prompt selecting k ICL examples. The batched generation method simultaneously performs text generation tasks on n input data.

By using batched generation, we can reduce the average token usage per instance by $\frac{n-1}{n}(token_{ins} + m \cdot token_{ic})$, where $token_{ins}$ and $token_{ic}$ respectively represent the number of

Table 1. Instruction and data format in prompt text

Dataset	Prompt text
E2E	Put the highlighted table together to form a sentence
	name : The Vaults | food : Japanese | price : cheap | family friendly:yes
WebNLG	Put the highlighted triples together to form a sentence
	Angola_International_Airport | 1st_runway_Number | 5
DART	Put the highlighted triples together to form a sentence
	Allan Shivers | ACTIVE_YEARS_START_DATE | 1947-01-21
ToTTo	Put the highlighted table together to form a sentence
	<page_title> Malcolm McDowell <section_title> Music video <table> <cell> 2009 <col_header> Year

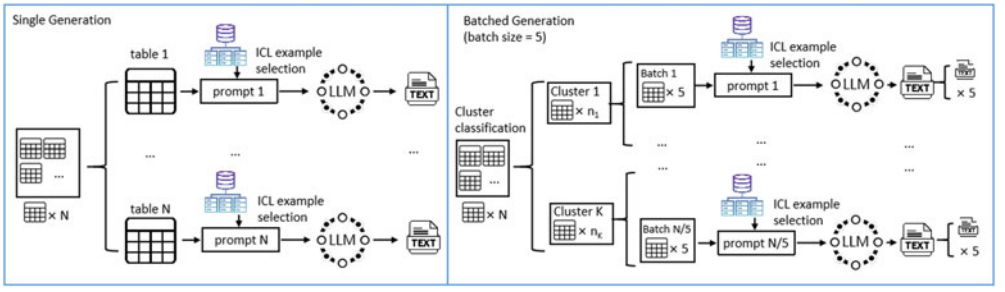

Figure 3. Left: **Single Generation**, where text is generated for one structural data at a time. Right: **Batched Generation**, with text being generated for 5 simultaneously.

Put the highlighted table/triples together to form a sentence:

Input :{ICL example data 1}

Output :{ICL example text 1}

...

Input :{ICL example data k}

Output :{ICL example text k}

Input :{input data}

Figure 4. Prompt for single generation.

tokens used for the instruction and a single in-context learning example within the prompt, n is the batch size, and m is the number of in-context learning examples. We provide detailed proofs in the subsequent sections.

3.5 Computational complexity analysis

Let M represent the number of samples in the training set, K denote the number of clusters in the first clustering, and m be the number of subclusters in the second clustering.

Preprocessing phase: The typical complexity of K-means clustering during the first phase of clustering is $O(MKDI_1)$, where I_1 is the number of iterations required for the algorithm to converge. Similarly, for the second phase of clustering within each primary cluster, the average complexity would be $O(\frac{M}{K}mDI_2)$, where I_2 is the number of iterations for the second round

Put the highlighted table/triples together to form a sentence:

Demonstration:

Input 1 :{ICL example data 1}

...

Input k :{ICL example data k}

Output 1 :{ICL example text 1}

...

Output k :{ICL example text k}

Next, Generate the text corresponding to the following n tables, and the format is the same as Demonstration.

Input 1 :{input data 1}

...

Input n :{input data n}

Figure 5. Prompt for batched generation.

of clustering to converge. The total complexity of the clustering phase can be represented as $O(M(m+K)D(I_1+I_2))$.

Inference phase: For each input sample, the algorithm compares it with K cluster centers to select the nearest one. The time complexity for this step would be $O(KD)$. Assume N be the number of samples in testing set, the complexity of inference phase is $O(KDN)$.

Considering K, I_1, I_2, and m as constants, and both M and N are significantly larger than K, I_1, I_2, and m, the overall algorithmic complexity of the DCCS method is thus $O((M+N)D)$.

Existing methods based on selecting similar samples (i.e., KATE Liu *et al.* 2022b and CEIL Ye *et al.* 2023) are designed to compare distances with all N samples during each inference. The complexity of these methods is $O(MND)$.

In scenarios where N is considerably large, the DCCS method can significantly decrease the time required for in-context example selection.

For DCCS-Batch, since DCCS-Batch and DCCS use the same ICL example selection method, the time for generating prompts is also similar. We constructed K queues (K represents the number of clusters), and after determining which cluster a test data belongs to, we stored it in this queue. When a queue is full (containing batch-size data), we combined them into a prompt (as shown in Figure 5), and then cleared the queue. This results in an additional table storage and retrieval process for DCCS batch compared to single-input DCCS, but the time required for similarity comparison between samples is negligible.

3.6 Token utilization efficiency in batched generation

For data-to-text tasks, let's assume m in-context learning samples are selected, n is the batch size. The final input prompt text to the LLM typically consists of the instruction, context, and test input. Assuming $token_{ins}$ and $token_{ic}$ respectively represent the number of tokens used for the instruction and a single in-context learning sample within the prompt. $token_i$ is the number of tokens used for the input data. For a traditional single inference prompt, the prompt length (L_P) is given by $token_{ins} + m \cdot token_{ic} + token_i$.

The input token per sample is calculated as:

$$token_a = \frac{1}{N}\sum_{i=1}^{N}(token_{ins} + m \cdot token_{ic} + token_i) \tag{11}$$

Table 2. Comparison of Retrieval Time Saved and Token Saved (%) between DCCS and KATE across datasets

		DCCS vs KATE		DCCS-Batch (5) vs KATE		DCCS-Batch (10) vs KATE	
Dataset	Train size	Time saved (%)	Token saved	Time saved (%)	Token saved (%)	Time saved (%)	Token saved (%)
E2E	4862	43.97	-	52.24	66.38	52.89	80.11
WebNLG	6940	53.61	-	61.88	67.57	62.13	76.41
DART	30,526	82.18	-	85.51	67.54	85.94	76.32
ToTTo	120,761	94.90	-	95.83	60.42	96.02	71.13

In contrast, for batched inference prompt,assumed batch size is n , the prompt length (L_P) is given by $token_{ins} + m \cdot token_{ic} + ntoken_i$. The input token per sample is calculated as:

$$token_b = \frac{1}{N} \sum_{i=1}^{N/n} (token_{ins} + m \cdot token_{ic} + token_i) \tag{12}$$

$$token_b - token_a = \frac{n-1}{n} (token_{ins} + m \cdot token_{ic}) \tag{13}$$

Assuming the instruction part of the prompt remains unchanged, as the batch size and the number of ICL samples increase, the token utilization efficiency of batched generation becomes higher. Table 2 presents the percentage reduction in retrieval time and token usage achieved by our DCCS, DCCS-Batch (batch size 5), and DCCS-Batch (batch size 10) methods in comparison to KATE, evaluated across four datasets of varying sizes.

4. Experiments

In this section, we present experiments designed to assess the DCCS and DCCS-Batch methods in terms of generation quality and efficiency.

4.1 Experimental setup

4.1.1 Datasets and environment

We conducted experiments on four widely used datasets for the data-to-text task: E2E (Novikova, Dušek, and Rieser 2017; Dušek et al. 2020), WebNLG (Gardent *et al.* 2017), DART (Nan *et al.* 2021), and ToTTo (Parikh *et al.* 2020).

E2E contains 42 000 restaurant meaning representations with up to eight slot–value pairs (e.g., *name*, *food*, *area*). Inputs are short and domain-specific, and most references are single sentences, making it a compact closed-domain benchmark.

WebNLG covers 15 DBpedia categories (e.g., *Airport*, *University*) and provides sets of 1–6 triples as input. Compared with E2E, it spans multiple domains and often requires multi-sentence outputs to verbalize more varied predicates.

DART extends WebNLG to an open-domain setting by mining RDF-style triples from Wikipedia tables and sentences. Its inputs are longer and noisier, so systems must generalize to unseen predicates and cope with imperfect schemas.

ToTTo is a table-to-text dataset that highlights a subset of Wikipedia table cells to be verbalized. Unlike the pure triple formats of WebNLG and DART, ToTTo inputs preserve table structure and require the model to reason over cell context and aggregation.

The experimental environment incorporated the following hardware configurations:

- CPU: Intel Xeon Gold 6240 CPU (2.6 GHz, 72 cores);
- RAM: 512 GB DDR4;
- GPU: NVIDIA Tesla T4.

We used the GPT-3.5 API (gpt-3.5-turbo-0125) provided by OpenAI[a] and the GLM-3 API (glm-3-turbo) provided by ZHIPU AI[b] as LLM in our method. We set the temperature parameter to 0 for GPT-3.5 and 0.01 for GLM-3, using the default configuration for other arguments. To minimize the influence of external factors on the in-context examples in the prompt, we adopted the method used by (Li *et al.* 2024), which involves merely inserting a brief instruction at the beginning of each prompt. For example, "Put the highlighted table/triples together to form a sentence." Detailed examples of such prompts can be seen in Table 1, Figure 4, and Figure 5. Considering the cost implications of LLM API calls, we conducted experiments using the full test dataset for batch generation methods. For single-generation methods with GPT-3.5 and GLM-3, we randomly selected **100** test samples, ensuring the same samples were used across all methods. Additionally, we conducted experiments with a smaller open-source model, LLaMA 3.1-8B,[c] which extends the context length to 128K and supports our long-prompt experiments. We have made the complete source code publicly available – including our DCCS implementation, all baseline methods, and the exact experimental configurations.[d]

4.1.2 Baseline

To evaluate the performance of our **DCCS** and **DCCS-Batch** methods, we benchmarked them against seven existing in-context example selection methodologies:

Random: For each test input, m samples are randomly selected from the training set to serve as in-context examples.

KATE (Liu *et al.* 2022b): The Knn-Augmented in-conText Example selection (KATE) method identifies the m training samples with the shortest distance to the test input in the embedding space.

KFN: In contrast to KATE, the K-Furthest Neighbors (KFN) method selects the m training samples that exhibit the longest distance to the test input in the embedding space.

DPP (Kulesza and Taskar 2011): The Determinantal Point Process (DPP) is designed for set selection problems emphasizing diversity. Following Ye *et al.* (2023) we first retrieve the top-100 nearest neighbors of the test instance with a dense KNN retriever, and then apply the conditional DPP MAP solver to pick m demonstrations that maximize a relevance–diversity objective.

BM25 (Karpukhin *et al.* 2020): BM25 is a widely used text retrieval method based on the term frequency-inverse document frequency (TF-IDF) principle. It ranks training samples according to their relevance to the test input, determined by a weighted TF-IDF scheme.

DPR (Karpukhin *et al.* 2020): The Dense Passage Retriever (DPR) employs a dense encoder that transforms text into d-dimensional vectors in dense space. It builds an index of all training passages for efficient retrieval. At runtime, a separate encoder converts the test input into a d-dimensional vector, retrieving the top-m passages whose vectors are most similar to the test input vector.

Random-Batch: This method randomly selects input samples to form a batch and chooses m samples from the training set as in-context examples in a random manner.

[a] https://platform.openai.com/
[b] https://open.bigmodel.cn/dev/api#glm-3-turbo
[c] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
[d] https://github.com/gerontech-hfut/InC-learning/tree/icl_data2text

Table 3. Comparative results of data-to-text generation using GPT-3.5 on the E2E, DART, and WebNLG (100 test samples) in a 5-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	E2E			DART			WebNLG		
Method	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
w/o example	36.11	41.72	63.56	30.66	44.44	65.93	40.03	53.09	68.78
Random	39.72	42.31	65.20	31.27	45.97	66.66	41.82	56.41	72.35
KATE	41.49	43.91	66.56	**34.30**	46.44	67.32	45.06	**57.47**	**73.04**
KFN	38.04	43.68	65.82	29.90	43.84	65.24	41.42	55.63	71.74
DPP	39.19	45.08	66.32	33.53	46.97	66.64	43.13	55.56	71.95
BM25	39.49	43.46	65.87	33.01	**47.68**	66.88	41.73	54.35	71.33
DPR	38.22	42.67	65.41	32.37	47.35	67.63	43.35	55.31	71.56
DCCS	**43.03**	**45.72**	**67.56**	33.76	47.60	**68.34**	**46.67**	57.21	72.61

4.1.3 Evaluation metrics

We evaluated the generated texts using four widely used automatic metrics: **BLEU** (Papineni *et al.* 2002), **ROUGE-L** (Lin 2004), and **BERTScore** (Zhang *et al.* 2019). For the ToTTo dataset, we used the official evaluation script[e] to calculate **BLEU** and **PARENT** (Dhingra *et al.* 2019). These metrics capture complementary aspects of generation quality, ranging from lexical overlap to deeper semantic and factual accuracy.

BLEU quantifies the similarity between a generated output and one or more reference texts by computing a geometric mean of modified n-gram precisions, adjusted with a brevity penalty (BP) to discourage trivially short outputs.

ROUGE-L focuses on the longest sequence of tokens that appear in both texts in the same order, though not necessarily contiguously. This metric effectively captures sentence-level fluency and coherence, outperforming simple n-gram matching.

BERTScore employs contextualized embeddings (e.g., from BERT) to measure semantic similarity. Each token in the generated text is matched to the most similar token in the reference text based on cosine similarity.

PARENT is designed for data-to-text generation, incorporating both lexical overlap and alignment with source data. It assigns credit for n-grams in the generated text that appear in the reference or can be inferred from the input data. By jointly considering textual similarity and factual consistency, PARENT better captures the factual consistency of the generated output with the provided information.

4.2 Automatic evaluation results

Tables 3 and 4 report the 5-shot single-generation results for GPT-3.5 across all automatic metrics. Table 5 provides the corresponding scores for GLM-3, and Tables 6 and 7 list the results for Llama-3.1-8B. Batched-generation results appear in Tables 8, 9, and 10. The analogous 10-shot tables are included in Appendix A and B.

4.2.1 Single generation performance

As shown in Tables 3–6, and 7, both our proposed method and the KATE method significantly outperform the Random baseline. The corresponding significance tests are reported in Section 4.2.5.

[e] https://github.com/google-research/language/blob/master/language/totto/totto_eval.sh

Table 4. Comparative results of data-to-text generation using GPT-3.5 on the ToTTo (100 test samples) in a 5-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	Overall		Overlap Subset		Nonoverlap Subset	
Method	BLEU	PARENT	BLEU	PARENT	BLEU	PARENT
w/o example	25.4	46.0	26.7	46.7	23.9	45.1
Random	26.2	50.1	28.8	50.7	23.9	49.5
KATE	**37.5**	**58.5**	**42.6**	**62.0**	**31.1**	**54.2**
KFN	23.2	44.5	22.5	42.3	23.9	47.3
DPP	27.1	50.2	28.1	49.2	26.0	51.6
BM25	33.8	53.3	41.3	57.1	25.4	48.5
DPR	34.6	55.5	41.3	59.5	27.0	50.5
DCCS	35.4	55.4	40.6	58.4	29.3	51.7

Table 5. Comparative results of data-to-text generation using GLM-3 on the E2E, DART, and WebNLG (100 test samples) in a 5-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	E2E			DART			WebNLG		
Method	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
Random	52.82	50.78	68.56	47.08	57.83	73.40	60.02	67.73	79.57
KATE	**55.29**	50.81	69.17	49.74	**60.65**	**75.01**	61.36	69.00	79.61
KFN	53.60	49.96	68.32	46.63	56.80	72.48	59.88	67.67	78.48
DPP	54.14	50.38	69.03	47.64	57.50	73.48	58.41	67.87	78.77
BM25	53.64	50.87	69.15	46.93	57.53	73.74	60.08	**69.17**	79.20
DPR	54.11	51.05	68.46	49.23	59.48	74.49	57.60	66.07	72.67
DCCS	54.41	**52.10**	**70.15**	**49.74**	60.23	74.61	**62.14**	68.89	**79.83**

Table 6. Comparative results of data-to-text generation using Llama-3.1 on the E2E, DART, and WebNLG in a 5-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	E2E			DART			WebNLG		
Method	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
w/o example	44.45	46.55	63.16	33.23	48.72	61.17	39.11	53.93	66.26
Random	45.48	46.99	64.69	35.26	49.61	65.74	44.75	56.61	68.47
KATE	49.17	49.04	68.33	37.17	50.67	66.25	46.67	57.64	71.45
KFN	42.99	45.37	62.38	34.30	48.94	64.15	36.44	51.24	62.12
DPP	47.45	48.52	65.40	36.36	50.20	65.87	46.16	57.47	71.35
BM25	46.50	48.39	64.56	36.51	50.45	65.92	46.87	58.03	69.66
DPR	45.70	47.50	64.64	34.21	47.85	60.39	45.16	57.23	70.65
DCCS	**54.67**	**53.73**	**69.02**	**37.42**	**51.05**	**67.72**	**48.00**	**58.23**	**71.55**

Table 7. BLEU and PARENT on the ToTTo using Llama-3.1 in a 5-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	Overall		Overlap Subset		Nonoverlap Subset	
Method	BLEU	PARENT	BLEU	PARENT	BLEU	PARENT
w/o example	24.3	48.18	28.1	49.99	21.1	46.39
Random	27.7	48.35	32.2	50.29	23.9	46.45
KATE	40.2	55.69	49.5	60.31	32.0	51.14
KFN	27.3	46.39	31.3	48.19	23.8	44.62
DPP	28.2	47.05	34.2	49.11	23.2	45.02
BM25	34.8	53.73	44.3	58.13	26.8	49.40
DPR	31.4	49.48	38.1	52.74	25.7	46.27
DCCS	**41.4**	**56.02**	**51.3**	**60.46**	**32.7**	**51.65**

Table 8. BLEU for batched generation on the E2E, WebNLG, DART and ToTTo using GPT-3.5 in a 5-shot setting. The best performance per dataset is shown in **bold**

Method	Batch Size	E2E	DART	WebNLG	ToTTo
Random-Batch	5	46.4	23.5	26.0	22.5
	10	42.4	23.5	15.3	17.0
Data-based Centroid	5	47.5	35.3	42.6	26.5
	10	46.9	32.7	36.0	25.6
Text-based Centroid	5	50.7	35.9	40.9	25.2
	10	48.2	35.6	38.0	25.0
DCCS-Batch	5	**53.2**	**37.6**	**44.5**	**30.2**
	10	**48.9**	**37.2**	**42.7**	**29.1**

Table 9. BLEU for batched generation on the E2E, WebNLG, DART and ToTTo using GLM-3 in a 5-shot setting. The best performance per dataset is shown in **bold**

Method	Batch size	E2E	DART	WebNLG	ToTTo
Random-Batch	10	51.0	36.6	45.6	24.4
DCCS-Batch	10	**53.5**	**37.9**	**47.6**	**28.9**

This demonstrates that, compared to randomly selecting in-context examples, selecting semantically similar examples tailored to each input instance more effectively enhances the generation quality of LLMs.

Random underperforms DCCS on all datasets, yet it consistently outperforms the zero-shot setting and occasionally matches the weaker similarity-based baselines. Uniform random sampling of m demonstrations still provides representative lexical and syntactic patterns, giving it helpful prior clues that it never gets in a zero-shot prompt. Because it requires neither similarity scoring nor index lookup, Random incurs negligible computational overhead (Table 11). Accordingly, when low-latency inference is essential and accurate retrieval is unavailable, Random offers a fast, competitive baseline, albeit still inferior to semantically informed selection.

Table 10. BLEU scores for batched generation on the E2E, DART, WebNLG, and ToTTo using Llama-3.1 in a 5-shot setting. The best performance per dataset is shown in **bold**

Method	Batch size	E2E	DART	WebNLG	ToTTo
Random-Batch	5	42.0	34.1	42.6	26.3
	10	38.7	33.6	42.1	27.3
Data-based Centroid	5	47.5	30.3	43.1	27.8
	10	46.4	30.5	43.1	29.6
Text-based Centroid	5	46.7	34.4	43.0	22.7
	10	47.5	34.7	41.3	28.8
DCCS-Batch	5	**50.1**	**36.6**	**44.4**	**29.4**
	10	**49.4**	**35.6**	**44.2**	**29.8**

Table 11. Comparison of 5-shot prompt generation time (ms)

Method	E2E	WebNLG	DART	ToTTo
Random	0.01	0.02	0.02	0.02
KATE	42.55	52.10	137.01	479.65
KFN	44.15	52.19	140.10	445.90
DPP	56.91	70.63	433.60	725.35
DPR	29.01	33.74	105.82	376.15
BM25	28.06	19.81	119.14	1615.94
DCCS	23.84	24.17	24.41	24.45
DCCS-batch	20.32	19.86	19.85	20.02

Our DCCS method consistently ranks first or second across most datasets and evaluation metrics, underscoring its strong generalization ability and adaptability to various dataset structures and evaluation criteria. In the E2E dataset, DCCS achieved the highest BERTScore (67.56, 70.15, and 69.02 for GPT-3.5, GLM-3, and LLaMA-3.1, respectively) in the 5-shot setting. On the WebNLG dataset, DCCS demonstrated robust adaptability to diverse and complex domains, attaining the highest BLEU scores across all three models under the same setting.

Our method performs slightly less effectively than KATE on certain more complex datasets, such as ToTTo in the GPT-3.5 experiments (Table 4) and DART in the GLM-3 experiments (Table 5). We attribute this to two main factors. First, compared to other datasets, ToTTo contains more complex textual separation tags in its training samples, which adversely affect the performance of RoBERTa as a semantic encoder – a limitation we plan to address in future work. Second, the ToTTo test set contains a substantial number of instances that are nearly identical to those in the training set. Leveraging a K-nearest neighbor strategy over the entire training set, the KATE method can retrieve highly similar examples, resulting in higher BLEU and related evaluation scores.

Although our method does not select the globally most similar samples due to its design, it provides notable improvements in computational efficiency – particularly on large-scale datasets such as ToTTo. Moreover, the two-stage clustering procedure in DCCS allows the same in-context examples to be reused for similar test instances, thereby enabling efficient batched generation. Subsequent experiments confirm that this design contributes to improvements in both generation accuracy and efficiency.

Table 12. Comparative results of data-to-text generation using Llama-3.1 on the E2E, DART, and WebNLG datasets in a 5-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

		E2E			DART			WebNLG		
Method	k-shot	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
Data-based	5	47.50	48.06	64.72	35.11	48.72	66.12	45.54	57.76	69.87
Centroid	10	48.02	48.74	65.35	36.37	50.99	66.76	45.75	57.66	71.38
Text-based	5	40.22	43.42	64.28	31.81	43.59	61.69	45.02	57.39	71.30
Centroid	10	46.14	46.88	66.56	34.74	48.21	65.14	45.63	57.52	71.47
Nearest	5	44.74	46.72	63.52	35.98	50.15	66.17	45.90	57.42	71.16
Cluster	10	48.00	48.32	65.89	36.31	50.63	66.28	46.62	58.10	71.77
DCCS	5	**54.67**	**53.73**	**69.02**	**37.42**	**51.05**	**67.72**	**48.00**	**58.23**	**71.55**
	10	**54.20**	**53.68**	**68.73**	**37.59**	**51.12**	**66.84**	**49.59**	**59.51**	**72.86**

4.2.2 Comparison with SOTA models

We compare our DCCS against the SOTA on each benchmark dataset. The BLEU scores of these state-of-the-art (SOTA) systems are directly cited from the original publications. For the WebNLG dataset, the highest reported BLEU score is 67.32, achieved by the Controlled Prefixing approach using a fine-tuned T5 model (Clive *et al.* 2022). On the E2E dataset, ALoRA attains 70.6 BLEU (Liu *et al.* 2024), while the UniD2T system sets the benchmark on ToTTo and DART with 49.9 and 54.96 BLEU, respectively (Li *et al.* 2024).

Fine-tuned models consistently outperform in-context learning (ICL) because they update millions of task-specific parameters, effectively absorbing corpus-level lexical and syntactic regularities. ICL, by design, keeps the backbone frozen and can only leverage a handful of demonstrations, so a performance gap is expected. Despite this, we pursue ICL for three pragmatic reasons: (i) zero training cost – new domains can be served without GPU-intensive fine-tuning; (ii) data frugality – no large parallel corpus is required, which is crucial for low-resource tables; and (iii) fast iteration – example sets can be swapped on-the-fly to meet downstream constraints (e.g., privacy or domain drift) that would otherwise force a costly model retrain.

We also find that GPT-3.5 and GPT-4, although strong in instruction following, occasionally inject polite or conversational fillers unrelated to the table, depressing automatic metrics. In contrast, LLaMA 3.1-8B, when prompted with the same demonstrations, tends to generate more concise, schema-aligned sentences and therefore yields higher BLEU under the ICL regime.

4.2.3 Ablation experiments

To evaluate the individual contributions of each component in our two-stage clustering framework, we conduct ablation studies by comparing the proposed DCCS method with several simplified variants.

Nearest Cluster (DCCS w/o second clustering): This variant employs only a single-stage clustering process on the training set. For each test input, the nearest cluster is identified by computing its similarity to all cluster centers. Rather than selecting the centroid, m in-context examples are randomly sampled from within the nearest cluster. This design helps isolate the impact of the second-stage refinement employed in our full method.

Diversity-based Clustering (Zhang *et al.* 2022): This baseline focuses solely on the diversity of in-context examples, irrespective of their semantic proximity to the test input. In the context of data-to-text generation, diversity is examined from two perspectives:

Table 13. BLEU and PARENT on the ToTTo using Llama-3.1 in a 5-shot setting. The best performance per metric is shown in **bold**

		Overall		Overlap Subset		Nonoverlap subset	
Method	k-shot	BLEU	PARENT	BLEU	PARENT	BLEU	PARENT
Data-based Centroid	5	28.0	49.12	31.0	50.59	25.2	47.67
	10	27.0	45.57	32.9	49.31	22.0	41.89
Text-based Centroid	5	27.3	44.73	31.9	45.73	23.3	43.74
	10	17.0	33.18	22.1	37.35	13.0	29.08
Nearest Cluster	5	26.5	47.53	29.8	49.13	23.6	45.96
	10	27.3	47.47	31.8	49.44	23.6	45.53
DCCS	5	**41.4**	**56.02**	**51.3**	**60.46**	**32.7**	**51.65**
	10	**41.9**	**57.41**	**51.6**	**61.80**	**33.3**	**53.08**

- **Data-based Centroid (DCCS w/o second clustering)**: Structured data entries from the training set are embedded and clustered into m groups using the K-Means algorithm. The centroids of these clusters are then selected as in-context examples.
- **Text-based Centroid (DCCS w/o first clustering)**: Similarly, the textual outputs associated with the training samples are embedded and clustered into m groups, with each cluster's centroid used as an in-context example.

As shown in Tables 12 and 13, under the same low-time-complexity constraint, approaches that consider only similarity (Nearest Cluster) or only diversity (Data-based Centroid and Text-based Centroid) score lower than our DCCS method on every dataset. A similarity-only strategy fails to ensure adequate semantic coverage of the example space, leading to many redundant in-context demonstrations. In contrast, a diversity-only strategy selects demonstrations that are semantically too distant from the test input, so the language model learns little beyond the desired output format.

4.2.4 Batched-generation performance

Tables 8, 9, and 10 report BLEU scores for different batched generation strategies across the E2E, DART, WebNLG, and ToTTo datasets, evaluated with GPT-3.5, GLM-3, and LLaMA-3.1. The proposed DCCS-Batch method consistently outperforms baseline approaches, demonstrating robust effectiveness in enhancing data-to-text generation quality.

For GPT-3.5, DCCS-Batch consistently surpasses other methods. On the E2E dataset, it achieves a BLEU score of 53.2 with a batch size of 5 – substantially outperforming the Random-Batch baseline (46.4). Similarly, it obtains scores of 44.5 on WebNLG and 30.7 on ToTTo, highlighting its ability to maintain output accuracy while controlling prompt length. Across GLM-3 and LLaMA-3.1, DCCS-Batch remains competitive, consistently outperforming both Text-based and Data-based Centroid baselines under both 5-shot and 10-shot configurations. These findings indicate that in batched generation scenarios, our method achieves more accurate outputs than both random selection and diversity-only strategies.

Overall, DCCS-Batch achieves the highest BLEU scores in most evaluation settings. By effectively balancing semantic similarity and textual diversity, it optimizes token efficiency while preserving high-quality generation.

4.2.5 Statistical analysis

Table 14 shows that, in the 5-shot GPT-3.5 setting, our DCCS method significantly outperforms the Random baseline ($p < 0.05$) on BLEU, ROUGE-L, and BERTScore across the E2E, DART, and

Table 14. Statistical comparison of the DCCS and Random methods on E2E, DART, and WebNLG datasets in 5-shot setting using GPT-3.5. Metrics include BLEU, ROUGE-L, and BERTScore. The t-test results (t-value and *p*-value) indicate that DCCS significantly outperforms Random across most metrics ($p < 0.05$)

Dataset	Metric	*t*-value	*p*-value
E2E	BLEU	2.02	0.045
	ROUGE-L	3.29	0.001
	BERTScore	2.03	0.042
DART	BLEU	6.89	$<$0.001
	ROUGE-L	7.59	$<$0.001
	BERTScore	2.80	0.005
WebNLG	BLEU	2.21	0.027
	ROUGE-L	2.68	0.007
	BERTScore	2.94	0.004

WebNLG datasets. Even on the domain-specific and relatively homogeneous E2E corpus, DCCS achieves notable improvements, demonstrating its ability to retrieve semantically complementary demonstrations within a narrow domain. For the more heterogeneous DART and WebNLG datasets, DCCS yields the largest gains, often with $p < 0.001$, reflecting its effectiveness in selecting demonstrations that balance relevance and diversity.

We further conducted a one-way repeated-measures ANOVA followed by Tukey's HSD to compare the four selection strategies – DCCS, KATE, BM25, and Random – on each dataset. The main effect of *method* is significant for every dataset (minimum $F_{3,16} = 10.91$, $p < 0.001$, $\eta^2 = 0.26$). Tukey-HSD confirms that DCCS significantly outperforms Random on all datasets; however, its advantage over KATE is not statistically significant. The complete Tukey matrices are provided in Appendix Table C1.

4.2.6 Determine the number of clusters

We select the number of clusters for the first clustering in the DCCS method based on the number that maximizes the average silhouette coefficient. The silhouette coefficient measures how similar a sample is to its own cluster compared to other clusters, with higher values indicating more cohesive clustering. Therefore, a higher average silhouette coefficient suggests better clustering quality.

We computed the average silhouette coefficient for all training samples while varying the number of clusters from 5 to 20 for each dataset. Figure 6 presents the average silhouette coefficient across training samples for the E2E, DART, WebNLG, and ToTTo datasets.

Based on these comparisons, we selected the number of clusters with the highest silhouette coefficient for the first clustering step. Specifically, we set the number of clusters to 19 for E2E, 6 for DART, 8 for WebNLG, and 11 for ToTTo.

4.2.7 The role of in-context examples' similarity

We compare similarity-based example selection methods (e.g., KATE, BM25) with those emphasizing diversity (e.g., clustering by data or text) and observe that merely pursuing diversity does not consistently improve performance in data-to-text generation. As shown in Table 3, prioritizing semantic similarity provides stronger guidance for mapping structured data into coherent and faithful text. While diverse examples can theoretically broaden the model's generalization, their contribution is constrained if the examples are not closely aligned with the input data.

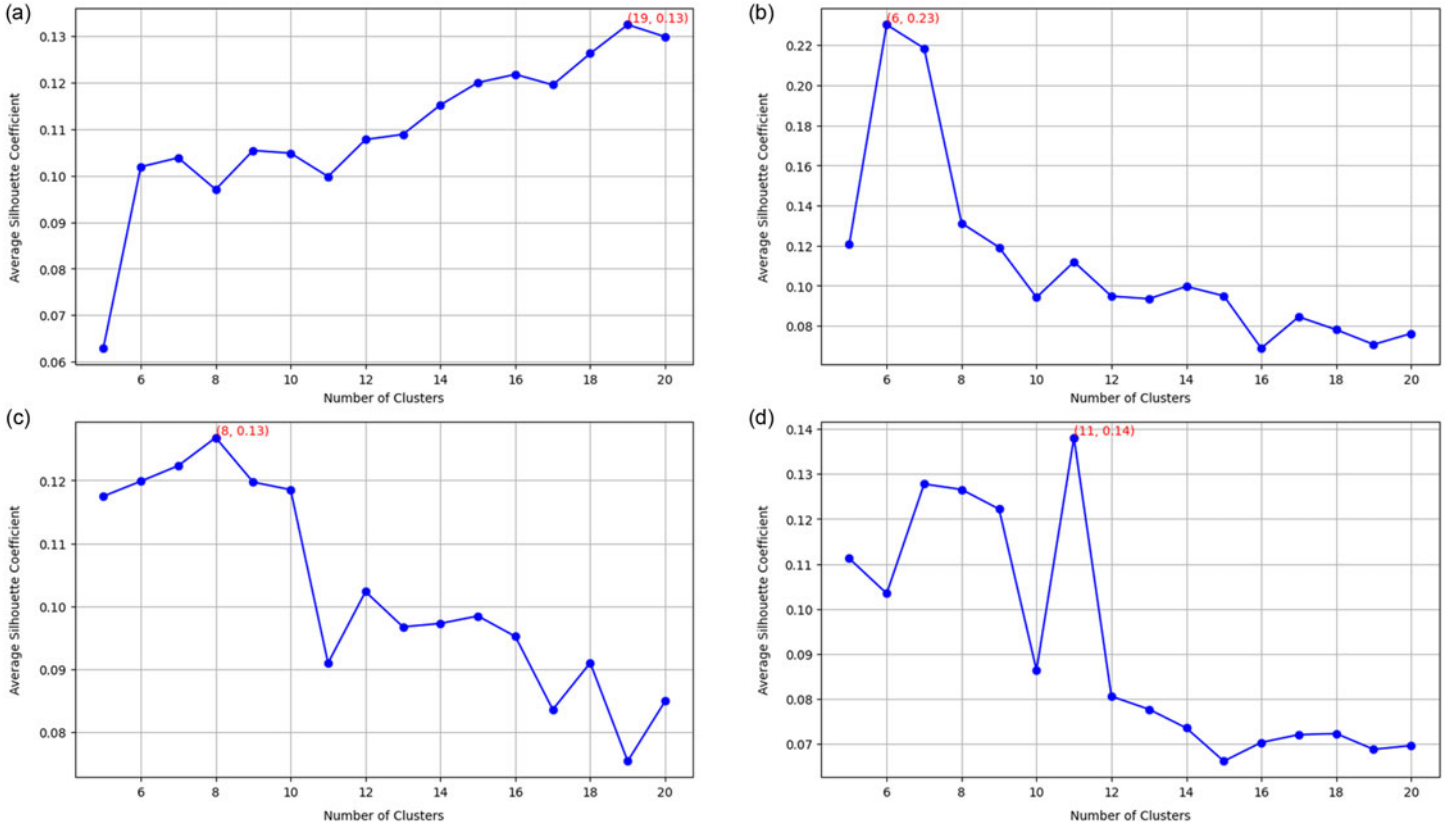

Figure 6. Average Silhouette Coefficient across varying cluster counts for the (a) E2E (top left), (b) DART (top right), (c) WebNLG (bottom left), (d) and ToTTo (bottom right) datasets. The number of clusters with the highest coefficient is chosen for the first clustering.

Nevertheless, our experiments also indicate that the highest possible similarity is not strictly necessary. Moderate relaxation of similarity criteria can still maintain content fidelity and textual quality while reducing computational costs. In particular, selecting a sufficiently similar yet not overly redundant set of examples helps retain the benefits of semantic alignment without inflating the inference load. This balance is demonstrated by our DCCS approach, which applies a two-stage clustering: first, we narrow down candidate examples by cluster centroids to cut down on exhaustive comparisons; second, we diversify the final selection of m in-context samples to cover a broader semantic space. By ensuring that each chosen example is both relevant and not excessively repetitive, we achieve robust generation performance with lower overhead.

4.2.8 Case study: the role of in-context examples' diversity

Table 15 presents a test example from the WebNLG dataset (5-shot setting) with three groups of in-context examples selected by Data-based Centroid, Nearest Cluster, and DCCS. Figure 7 visualizes the semantic distribution of these examples, where each text is encoded using RoBERTa-large and then reduced to two dimensions via PCA. Blue points represent examples selected by the DCCS method, green points by Data-based Centroid, gold points by Nearest Cluster, and red points indicate the reference text.

Compared to the Data-based Centroid method, our DCCS approach achieves a superior balance between diversity and relevance. Although the Data-based Centroid method selects cluster centers to maximize semantic diversity (as evidenced by the wider green distribution in Figure 7), this approach often positions cluster centers farther from the desired reference text (red points). Such misalignment can result in prompts containing in-context examples that fail to adequately capture the semantic content needed to guide the model toward generating outputs closely aligned with the reference text.

In contrast, the Nearest Cluster method improves the likelihood of selecting in-context examples that are closer to the reference text (red points), owing to its focus on relevant clusters. However, by randomly sampling in-context examples within a cluster, the method neglects semantic diversity, frequently resulting in examples with overlapping semantic coverage. This overlap diminishes the ability of the prompt to encompass a broad range of semantic contexts related to the reference text.

By incorporating a second clustering step, DCCS addresses the limitations of both approaches. It ensures relevance by focusing on clusters proximal to the test input while simultaneously enhancing diversity within the selected cluster. This dual-layered approach allows DCCS to generate prompts that both represent a wide semantic space and maintain strong alignment with the reference text, leading to improved generation quality.

4.3 Efficiency evaluation results

From the automated evaluation results, we observed significant improvements with our DCCS method compared to random and diversity-based approaches. While the accuracy improvement of our method over similarity-based methods such as KATE and BM25 was not as prominent (achieving second-best results in some cases), DCCS demonstrated substantial efficiency gains. These efficiency improvements are evident in two aspects: **Time Efficiency** and **Cost Efficiency**.

4.3.1 Time efficiency

Compared to existing methods that require comparisons with all samples in the training set, our method necessitates comparison with only K clustering centers, K is significantly smaller than the number of training set size. To demonstrate the efficiency improvement of our method in prompt generation, we compared the average time required to construct the prompt for each input to the

Table 15. A test sample with three groups of in-context examples selected by Data-based Centroid, Nearest Cluster, and DCCS from the WebNLG dataset in 5-shot setting

	Data	Text
Input data and gold reference	Tomato – family – Solanaceae Amatriciana_sauce – ingredient – Tomato Tomato – order – Solanales	Tomatoes which are part of the solanaceae family and solanales order are a key ingredient in Amatriciana sauce
Data-based centroid	103_Colmore_Row – architecturalStyle – Brutalist_architecture 103_Colmore_Row – location – Colmore_Row 103_Colmore_Row – architect – John_Madin John_Madin – birthPlace – Birmingham	103 Colmore Row is located at Colmore Row and designed by John Madin born in Birmingham. It has the Brutalist architecture style
	Alderney_Airport – runwayLength – 497.0 Alderney_Airport – 1st_runway_SurfaceType – Asphalt Alderney_Airport – cityServed – Alderney Alderney_Airport – elevationAboveTheSeaLevel_(in_metres) – 88.0 Alderney_Airport – runwayName – "03/21"	Alderney Airport serves the city of Alderney and situated 88 meters above sea level and its first runway is surfaced with asphalt.03/21 is the name of the airport's runway and it is 497.0 m long
	Alan_Bean – was a crew member of – Apollo_12 Alan_Bean – nationality – United_States Apollo_12 – operator – NASA Alan_Bean – occupation – Test_pilot Alan_Bean – birthPlace – Wheeler,_Texas Alan_Bean – almaMater – "UTAustin, B.S. 1955"	Alan Bean, an American was born in Wheeler Texas and graduated from UT Austin in 1955 with a BSc. He was a test pilot and a crew member on NASA's Apollo 12
	Batchoy – country – Philippines Batchoy – ingredient – Pork	Batchoy is eaten in the Philippines and one of the ingredients in it is pork
	A.C._Lumezzane – manager – Michele_Marcolini Michele_Marcolini – club – A.C._Chievo_Verona A.C._Lumezzane – ground – Italy Italy – leader – Pietro_Grasso Michele_Marcolini – club – Vicenza_Calcio	AC Lumezzane's ground is in Italy and their manager is Michele Marcolini who plays for A.C. Chievo Verona and is in the Vicenza Calcio Club. The leader of Italy is Pietro Grasso
Generated text	Tomatoes, featuring Brutalist architecture and located at 103 Colmore Row, are widely recognized for being an ingredient in Amatriciana sauce	
Nearest cluster	Binignit – region – Visayas Binignit– main Ingredients – Sago Binignit – country – Philippines Binignit – ingredient – Coconut_milk	Binignit is a dish from the region of Visayas, Phillippines. It includes coconut milk and sago. The binignit dish can be found in the Visayas region, Philippines. It includes sago and coconut milk
	Binignit – mainIngredients – Sweet_potato Sweet_potato – division – Flowering_plant Binignit – ingredient – Sago Sweet_potato – order – Solanales	Binignit includes the ingredients of sago and sweet potato, the latter of which is part of the order Solanales and is a flowering plant
	Indonesia – capital – Jakarta Bakso – region – Indonesia Bakso – country – Indonesia Indonesia – leaderName – Joko_Widodo	Bakso is food found in Indonesia, where Joko Widodo is the leader, and Jakarta is the capital
	Alderney_Airport – 1st_runway_SurfaceType – Poaceae Poaceae – order – Poales Poaceae – order – Commelinids	The surface of the 1st runway at Alderney airport is poaceae which belongs to the Poales and Commelinids orders
	Binignit – mainIngredients – Sweet_potato Sweet_potato – division – Flowering_plant Binignit – ingredient – Taro Sweet_potato – order – Solanales	Binignit's main ingredients are sweet potatoes which belongs to flowering plants and the Solanales order. The dish also contains Taro

Table 15. (Continued).

	Data	Text
Generated text	Tomatoes, belonging to the Solanaceae family and the Solanales order,are a key ingredient in Binignit	
DCCS	Bandeja_paisa – ingredient – Lemon Lemon – order – Sapindales Lemon – family – Rutaceae Bandeja_paisa – country – Colombian_cuisine Bandeja_paisa – region – Paisa_Region	The Colombian dish Bandeja paisa hails from the Palsa region. A main ingredient is the lemon which belongs to the Sapindales order of fruits,in the Rutaceae family
	Binignit – mainIngredients – Sweet_potato Sweet_potato – division – Flowering_plant Binignit – ingredient – Sago Sweet_potato – order – Solanales	Binignit includes the ingredients of sago and sweet potato, the latter of which is part of the order Solanales and is a flowering plant
	Amatriciana_sauce – ingredient – Tomato Tomato – order – Solanales	Tomato is an ingredient of Amatriciana sauce and is a member of the order Solanales
	Antioquia_Department – country – Colombia Avocado – order – Laurales Bandeja_paisa – ingredient – Avocado Avocado – family – Lauraceae Bandeja_paisa – region – Antioquia_Department	Avocado is a fruit of the Laurales order and Lauraceae family. It is an ingredient found in Bandeja paisa. That dish comes from Antioquia Department region in Columbia
	Tomato – family – Solanaceae Amatriciana_sauce – ingredient – Tomato	The tomato comes from the Solanaceae family and is an ingredient in Amatriciana sauce
Generated text	Tomato comes from the Solanaceae family and is classified under the Solanales order. It is an ingredient found in Amatriciana sauce	

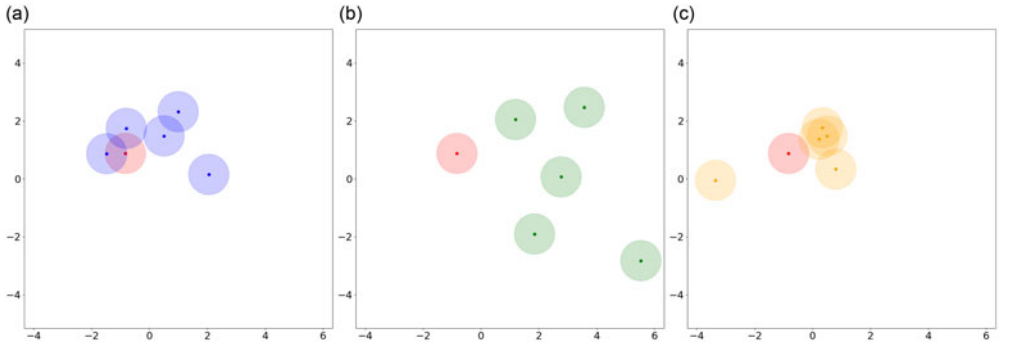

Figure 7. Semantic representation of in-context examples selected by different methods. (a) Blue points represent the DCCS method, (b) Green points represent the Data-based Centroid method, (c) Orange points represent the Nearest Cluster method, and Red points indicate reference text.

test across four datasets using different methods. We selected 5 in-context examples. The training datasets consist of the following sample sizes: 4,862 samples for the E2E dataset, 6,940 samples for the WebNLG dataset, 30,526 samples for the DART dataset, and 120,761 samples for the ToTTo dataset. Due to the need for embedding structural data whenever test data is received and then comparing these embeddings with those from the training set, a significant amount of time is consumed. Therefore, we pre-generated and saved the embedding results of the training set to optimize the process.

Table 11 shows that there is a positive correlation between the processing time of both KATE and DPP methods and the size of the dataset. In contrast, our method exhibits considerably more consistent and faster performance. The time cost is approximately 23.84, 24.17, 24.41, and 24.45

ms across different datasets. This consistency can be attributed to the fact that the largest time expenditure in prompt generation lies in the comparison of distances between samples. These experiments indicate that our DCCS method indeed manages to significantly reduce the time required to construct prompts for each input while ensuring the effectiveness of the selected in-context examples.

For frequency-based search method(e.g., BM25), although it does not incur the time cost of embedding the text, its performance significantly lags behind vector-based representations when dealing with large training datasets (such as the ToTTo dataset). This is because for each query term, the BM25 algorithm needs to search through all documents and calculate its contribution, which substantially increases the computation time as the size of the document collection grows.

DCCS-Batch requires slightly less time per sample compared to DCCS. This is because, even though both methods involve the same number of distance computations for each test input, DCCS must extract in-context examples from the training set for every individual test sample, whereas DCCS-Batch only needs to perform this extraction once per batch.

4.3.2 Cost efficiency

We measure efficiency in terms of the average number of tokens per instance (TpI), encompassing both input and output tokens. In principle, using larger batch sizes allows for higher token utilization because the fixed overhead of prompt tokens is amortized across multiple test samples. However, we observed in our experiments that excessively large batch sizes can cause the model to lose focus on individual test samples. This problem, which we call Generation Failure, occurs when the model fails to produce the correct number of sentences corresponding to the batch size or otherwise deviates from the required output.

Figure 8 presents the generation failure rate and token utilization rate at different batch sizes using the DCCS method. The results suggest that a batch size of 5–10 strikes a balance between low generation failure and efficient token usage. For example, on the WebNLG dataset, using a batch size of 5 reduced token usage to 32.43% of the single-generation baseline, while a batch size of 10 lowered it further to 23.59%. Under OpenAI's GPT-3.5 pricing as of May 21, 2024 ($0.0005 per 1K input tokens and $0.0015 per 1K output tokens for gpt-3.5-turbo-0125), these token savings translated into cost reductions of 60.85% and 68.84%, respectively.

4.4 Human evaluation results

In addition to automated evaluation, we conducted a human evaluation study on the E2E, DART, and WebNLG datasets. Following widely used text generation criteria (van der Lee *et al.* 2021; Fu *et al.* 2023), we adopted three assessment dimensions – **Fluency**, **Informativeness**, and **Relevance** – to evaluate the text generated by the LLMs. Below are the definitions of these dimensions (Fu *et al.* 2023):

- **Fluency**: Is the generated text well-written and grammatically correct?
- **Informativeness**: How well does the generated text capture the key ideas of the source data?
- **Relevance**: How closely is the generated text related to the source data?

We employed the **Categorical Choice** method to compare the DCCS method against KATE and Random. Empirical studies have shown that, in human evaluation tasks for text generation, asking participants to select the best or worst outcome is more reliable and consistent than using Likert scale-based scoring (van der Lee *et al.* 2021). For Fluency, participants directly compared the paired outputs for each test sample. For Informativeness and Relevance, participants were provided with the input data fed to the LLM before comparing the paired outputs for each test sample.

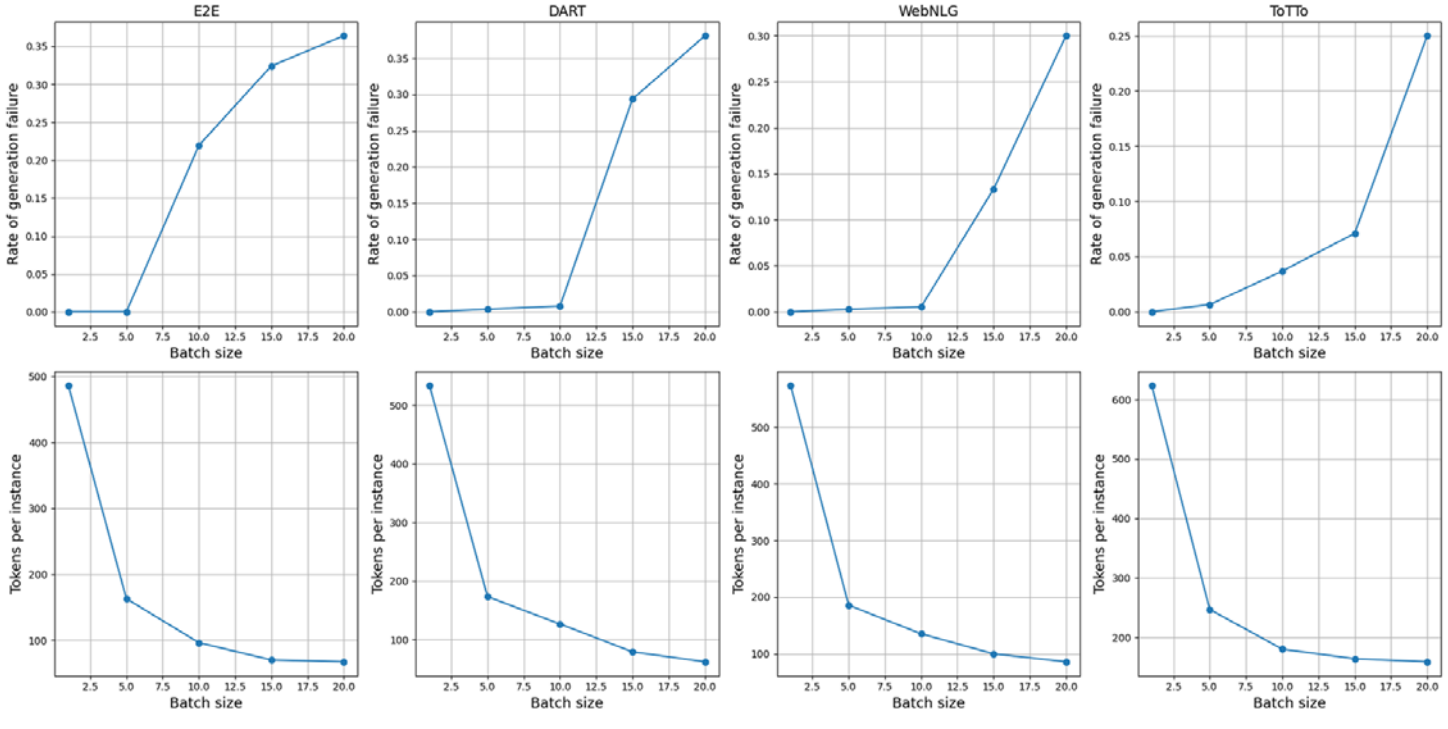

Figure 8. The change in Generation Failure Rate and the Average Number of Tokens per Instance (TpI) with increasing batch size.

Table 16. Average human evaluation scores (in percentage) for DCCS-generated outputs compared to KATE and Random across Fluency, Informativeness, and Relevance dimensions

Comparison	Dataset	Fluency (%)	Informativeness (%)	Relevance (%)
DCCS vs. Random	E2E	54.0	73.3	70.7
	DART	69.0	72.3	76.0
	WebNLG	72.3	73.7	76.7
DCCS vs. KATE	E2E	48.0	58.3	48.7
	DART	52.0	54.3	51.0
	WebNLG	51.0	57.7	57.0

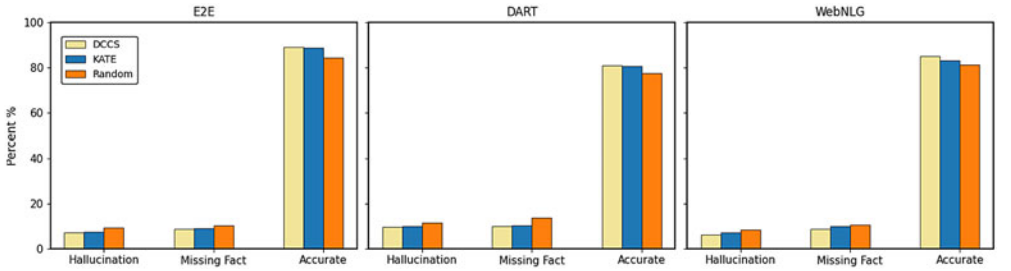

Figure 9. Human evaluation of factual consistency for GPT-3.5 outputs on the E2E, DART, and WebNLG datasets (Hallucination ↓, Missing Fact ↓, Accurate ↑).

All sentences were produced by GPT-3.5 using five in-context examples selected by the respective method. For each dataset, we created 200 sentence pairs, giving 600 comparisons overall. Six graduate-level NLP researchers – none otherwise involved in the project – independently chose the better sentence in every pair. Fleiss' κ across the six raters was 0.63, indicating substantial agreement.

Table 16 reports, for each criterion, the percentage of comparisons in which the DCCS output was preferred to the baseline (KATE or Random). DCCS is consistently favored over Random across all datasets and criteria, confirming the value of principled example selection. Compared with KATE, DCCS achieves higher preference rates on most metrics – particularly informativeness and relevance on the more heterogeneous DART and WebNLG sets – even though KATE requires substantially higher computation.

4.5 Factual consistency metrics

To complement the subjective criteria above, we also assessed *factual consistency*. Following Li *et al.* (2024), each output was checked for two non-exclusive errors – *Hallucination* and *Missing Fact*. A sentence is labeled *Accurate* only when neither error is present.

Six annotators, independent of the study, each labeled 100 sampled test instances per dataset. Source triples or table rows were displayed alongside the three candidate sentences. Annotators marked both error types, and *Accurate* was derived automatically as no Hallucination & no Missing Fact.

Figure 9 shows the proportion of sentences exhibiting each error. DCCS produces the fewest hallucinations and omissions across all datasets, yielding the highest share of accurate outputs. These findings echo the human-preference results in Table 16: the dual-clustering strategy not only enhances perceived *informativeness* and *relevance* but also markedly improves factual fidelity.

Table 17. Comparison of BLEU scores between fine-tuned state-of-the-art (SOTA) models and our DCCS-based in-context learning (ICL) method using 5-shot prompts across different base models and datasets

Method	Model	Training strategy	WebNLG	DART	E2E	ToTTo
Controlled prefixing	T5-large	Fine-tuning	**67.32**	51.95	44.1	—
UniD2T	T5-large	Fine-tuning	60.41	**54.96**	—	**49.9**
ALoRA	LlaMA-2 7B	PEFT	—	—	**70.6**	—
DCCS	GPT-4	ICL	46.89	36.91	52.1	31.9
DCCS	GPT-3.5	ICL	45.40	36.65	50.9	29.2
DCCS	llama-3.1 8B	ICL	48.00	37.42	54.7	41.4

5. Limitations and future work

Although our proposed in-context example-selection method improves accuracy, prompt-construction efficiency, and generation speed for data-to-text tasks, several limitations remain.

First, certain error types persist even under the DCCS framework. We still observe *missing facts*, where salient input fields are omitted; *hallucinations*, where information unsupported by the input is introduced; and *output-format errors*, which occur in batch generation, particularly when the batch size exceeds ten. Our analysis indicates that these errors become more pronounced when the training examples selected for a cluster differ structurally from the test instance (e.g., in field count or entity type). While dual clustering improves relevance and diversity, it does not fully guarantee factual completeness or consistency. Future work will explore incorporating chain-of-thought prompting and LLM self-revision to mitigate these issues.

Second, *anchoring* and *attribution* remain practical concerns. In our current benchmarks (WebNLG, E2E, DART, and ToTTo), each test instance contains fewer than ten triples or table rows, allowing us to directly present the raw input data. However, when the method is applied in larger-scale scenarios – such as enterprise reporting or knowledge-base verbalization – each generated fact should be explicitly paired with its provenance. Providing verifiable provenance enables readers to quickly validate each statement and correct any factual errors introduced by the model.

6. Conclusion

In this paper, we presented a method to enhance data-to-text generation using LLMs by focusing on the effective selection of in-context examples through our Double Clustering-based in-Context example Selection (DCCS) method. This method emphasizes the importance of example similarity and diversity, and, along with the proposed batched generation method, significantly improves efficiency and performance in text generation tasks.

Our experimental findings on various datasets demonstrate the potential of our approach to significantly improve LLM performance. This success opens avenues for further research into more personalized text generation and the application of our methods across a broader spectrum of tasks and models. By doing so, we aim to increase the utility and personalization of language model outputs for diverse user needs.

Future work could explore further optimization of the clustering process, the application of our method to other types of generation tasks, and the integration of additional contextual information to further enhance the personalization and relevance of generated outputs.

Acknowledgements. This work was partially supported by the National Natural Science Foundation of China (No. 62072153), the Anhui Provincial Key Technologies R&D Program (No. 2022h11020015), and the 111 Center (No. B14025).

References

Brown T., Mann B., Ryder N., Subbiah M., Kaplan J. D., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A. et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33* , pp. 1877–1901.

Chen M., Tworek J., Jun H., Yuan Q., Pinto H.P.d.O., Kaplan J., Edwards H., Burda Y., Joseph N., Brockman G. et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Clive J., Cao K. and Rei M. (2022). Control prefixes for parameter-efficient text generation. In Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM), pp. 363–382.

Deng L., Zhang S. and Balog K. (2019). Table2vec: Neural word and entity embeddings for table population and retrieval. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, pp. 1029–1032.

Dhingra B., Faruqui M., Parikh A., Chang M.-W., Das D. and Cohen W. (2019). Handling divergent reference texts when evaluating table-to-text generation. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy. Association for Computational Linguistics, pp. 4884–4895.

Dong Q., Li L., Dai D., Zheng C., Ma J., Li R., Xia H., Xu J., Wu Z., Chang B., Sun X., Li L. and Sui Z. (2024). A survey on in-context learning. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Miami, Florida, USA. Association for Computational Linguistics, pp. 1107–1128.

Dušek O., Novikova J. and Rieser V. (2020). Evaluating the state-of-the-art of end-to-end natural language generation: the e2e nlg challenge. *Computer Speech & Language* **59**, 123–156.

Fu J., Ng S.-K., Jiang Z. and Liu P. (2023). Gptscore: Evaluate as you desire.

Gardent C., Shimorina A., Narayan S. and Perez-Beltrachini L. (2017). The webnlg challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on Natural Language Generation, pp. 124–133.

Gatt A. and Krahmer E. (2018). Survey of the state of the art in natural language generation: core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* **61**, 65–170.

Gonen H., Iyer S., Blevins T., Smith N. and Zettlemoyer L. (2023). Demystifying prompts in language models via perplexity estimation. In **Bouamor H., Pino J. and Bali K.** (eds), *Findings of the Association for Computational Linguistics: EMNLP 2003, Singapore*. Association for Computational Linguistics, pp. 10136–10148.

Hallett C., Power R. and Scott D. (2006). Summarisation and visualisation of e-health data repositories. In UK E-Science All-Hands Meeting.

Herzig J., Müller T., Krichene S. and Eisenschlos J. (2021). Open domain question answering over tables via dense retrieval. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, pp. 512–519.

Karpukhin V., Oguz B., Min S., Lewis P., Wu L., Edunov S., Chen D. and Yih W.-t. (2020). Dense passage retrieval for open-domain question answering. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics.

Kaufman L. and Rousseeuw P.J. (2009). *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons.

Kulesza A. and Taskar B. (2011). k-dpps: fixed-size determinantal point processes. In Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, Madison, WI, USA. Omnipress, pp. 1193–1200.

Lebret R., Grangier D. and Auli M. (2016). Neural text generation from structured data with application to the biography domain. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 1203–1213.

Levy I., Bogin B. and Berant J. (2023). Diverse demonstrations improve in-context compositional generalization. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Toronto, Canada. Association for Computational Linguistics, pp. 1401–1422.

Li S., Li L., Geng R., Yang M., Li B., Yuan G., He W., Yuan S., Ma C., Huang F. et al. (2024). Unifying structured data as graph for data-to-text pre-training. *Transactions of the Association for Computational Linguistics* **12**, 210–228.

Li X., Lv K., Yan H., Lin T., Zhu W., Ni Y., Xie G., Wang X. and Qiu X. (2023). Unified demonstration retriever for in-context learning. arXiv preprint arXiv:2305.04320.

Li X.L. and Liang P. (2021). Prefix-tuning: optimizing continuous prompts for generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 4582–4597.

Lin C.-Y. (2004). Rouge: a package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81.

Liu A., Dong H., Okazaki N., Han S. and Zhang D. (2022a). PLOG: table-to-logic pretraining for logical table-to-text generation. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics, pp. 5531–5546.

Liu J., Shen D., Zhang Y., Dolan B., Carin L. and Chen W. (2022b). What makes good in-context examples for GPT-3? In Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, Dublin, Ireland. Association for Computational Linguistics, pp. 100–114.

Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L. and Stoyanov V. (2019). Roberta: a robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Liu Z., Lyn J., Zhu W., Tian X. and Graham Y. (2024). ALoRA: allocating low-rank adaptation for fine-tuning large language models. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Mexico City, Mexico. Association for Computational Linguistics, pp. 622–641.

Lu Y., Bartolo M., Moore A., Riedel S. and Stenetorp P. (2022). Fantastically ordered prompts and where to find them: overcoming few-shot prompt order sensitivity. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland. Association for Computational Linguistics, pp. 8086–8098.

Min S., Lyu X., Holtzman A., Artetxe M., Lewis M., Hajishirzi H. and Zettlemoyer L. (2022). Rethinking the role of demonstrations: what makes in-context learning work? arXiv preprint arXiv:2202.12837.

Nakano R., Hilton J., Balaji S., Wu J., Ouyang L., Kim C., Hesse C., Jain S., Kosaraju V., Saunders W. et al. (2021). Webgpt: browser-assisted question-answering with human feedback. arXiv preprint arXiv:2112.09332.

Nan L., Radev D., Zhang R., Rau A., Sivaprasad A., Hsieh C., Tang X., Vyas A., Verma N., Krishna P., Liu Y., Irwanto N., Pan J., Rahman F., Zaidi A., Mutuma M., Tarabar Y., Gupta A., Yu T., Tan Y.C., Lin X.V., Xiong C., Socher R. and Rajani N.F. (2021). DART: Open-domain structured data record to text generation. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Online. Association for Computational Linguistics, pp. 432–447.

Novikova J., Dušek O. and Rieser V. (2017). The E2E dataset: new challenges for end-to-end generation. In Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany. Association for Computational Linguistics, pp. 201–206.

OpenAI (2023). Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

Papineni K., Roukos S., Ward T. and Zhu W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics, pp. 311–318.

Parikh A., Wang X., Gehrmann S., Faruqui M., Dhingra B., Yang D. and Das D. (2020). Totto: a controlled table-to-text generation dataset. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1173–1186.

Qin C., Zhang A., Chen C., Dagar A. and Ye W. (2024). In-context learning with iterative demonstration selection. In **Al-Onaizan Y., Bansal M. and Chen Y.-N.** (eds), *Findings of the Association for Computational Linguistics: EMNLP, Miami, Florida, USA*. Association for Computational Linguistics, pp. 7441–7455.

Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. et al. (2019). Language models are unsupervised multitask learners. *OpenAI Blog* **1**(8), 9.

Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M., Zhou Y., Li W. and Liu P.J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* **21**(1), 5485–5551.

Rousseeuw P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* **20**, 53–65.

Rubin O., Herzig J. and Berant J. (2022). Learning to retrieve prompts for in-context learning. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Seattle, United States. Association for Computational Linguistics, pp. 2655–2671.

Sorensen T., Robinson J., Rytting C., Shaw A., Rogers K., Delorey A., Khalil M., Fulda N. and Wingate D. (2022). An information-theoretic approach to prompt engineering without ground truth labels. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland. Association for Computational Linguistics, pp. 819–862.

Su H., Kasai J., Wu C.H., Shi W., Wang T., Xin J., Zhang R., Ostendorf M., Zettlemoyer L., Smith N.A. et al. (2022). Selective annotation makes language models better few-shot learners. arXiv preprint arXiv:2209.01975.

Tanwar E., Dutta S., Borthakur M. and Chakraborty T. (2023). Multilingual LLMs are better cross-lingual in-context learners with alignment. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Toronto, Canada. Association for Computational Linguistics, pp. 6292–6307.

Turner R., Sripada S., Reiter E. and Davy I.P. (2008). Using spatial reference frames to generate grounded textual summaries of georeferenced data. In Proceedings of the Fifth International Natural Language Generation Conference, pp. 16–24.

van der Lee C., Gatt A., van Miltenburg E. and Krahmer E. (2021). Human evaluation of automatically generated text: current trends and best practice guidelines. *Computer Speech and Language* **67**, 101151.

Wang S., Chen Z., Shi C., Shen C. and Li J. (2024). Mixture of demonstrations for in-context learning. In **Globerson A., Mackey L., Belgrave D., Fan A., Paquet U., Tomczak J. and Zhang C.** (eds), *Advances in Neural Information Processing Systems*, vol. **37**. Curran Associates, Inc., pp. 88091–88116.

Wei J., Wang X., Schuurmans D., Bosma M., ichter b., Xia F., Chi E., Le Q.V. and Zhou D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In **Koyejo S., Mohamed S., Agarwal A., Belgrave D., Cho K. and Oh A.** (eds), *Advances in Neural Information Processing Systems*, vol. **35**. Curran Associates, Inc., pp. 24824–24837.

Wen T.-H., Gasic M., Mrkšić N., Su P.-H., Vandyke D. and Young S. (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1711–1721.

Wiseman S., **Shieber S. and Rush A.** (2017). Challenges in data-to-document generation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, pp. 2253–2263. Copenhagen, Denmark.

Wu Z., **Wang Y.**, **Ye J. and Kong L.** (2023a). Self-adaptive in-context learning: an information compression perspective for in-context example selection and ordering. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Toronto, Canada. Association for Computational Linguistics, pp. 1423–1436.

Wu Z., **Wang Y.**, **Ye J.**, **Wu Z.**, **Feng J.**, **Xu J. and Qiao Y.** (2023b). OpenICL: an open-source framework for in-context learning. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), Toronto, Canada. Association for Computational Linguistics, pp. 489–498.

Ye J., **Wu Z.**, **Feng J.**, **Yu T. and Kong L.** (2023). Compositional exemplars for in-context learning. In Proceedings of the 40th International Conference on Machine Learning, ICML'23. JMLR.org.

Yuan C. and Yang H. (2019). Research on k-value selection method of k-means clustering algorithm. *J* **2**(2), 226–235.

Zhang T., **Kishore V.**, **Wu F.**, **Weinberger K.Q. and Artzi Y.** (2019). Bertscore: evaluating text generation with bert. arXiv preprint arXiv:1904.09675.

Zhang Z., **Zhang A.**, **Li M. and Smola A.** (2022). Automatic chain of thought prompting in large language models.

Zhao Y., **Qi Z.**, **Nan L.**, **Flores L.J. and Radev D.** (2023a). LoFT: enhancing faithfulness and diversity for table-to-text generation via logic form control. In Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, Dubrovnik, Croatia. Association for Computational Linguistics, pp. 554–561.

Zhao Y., **Zhang H.**, **Si S.**, **Nan L.**, **Tang X. and Cohan A.** (2023b). Investigating table-to-text generation capabilities of large language models in real-world information seeking scenarios. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track, Singapore. Association for Computational Linguistics, pp. 160–175.

Zhao Y., **Zhang H.**, **Si S.**, **Nan L.**, **Tang X. and Cohan A.** (2023c). Large language models are effective table-to-text generators, evaluators, and feedback providers. arXiv preprint arXiv:2305.14987.

Zhao Z., **Wallace E.**, **Feng S.**, **Klein D. and Singh S.** (2021). Calibrate before use: Improving few-shot performance of language models. In International Conference on Machine Learning. PMLR, pp. 12697–12706.

Appendix A. Additional Single Generation Experiment Results

Table A1. Comparative results of data-to-text generation using GPT-3.5 on the E2E, DART, and WebNLG (100 test samples) in a 10-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	E2E			DART			WebNLG		
Method	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
w/o example	36.11	41.72	63.56	30.66	44.44	65.93	40.03	53.09	68.78
Random	39.09	42.45	65.68	31.02	44.64	66.05	44.55	56.88	72.87
KATE	43.31	**45.93**	**67.50**	**34.50**	46.56	67.38	**47.31**	57.81	73.89
KFN	37.65	41.65	64.49	27.05	43.55	65.08	41.92	55.79	72.42
DPP	41.61	45.51	65.27	32.48	46.90	67.56	44.57	57.77	73.32
BM25	41.84	45.18	66.65	33.64	**48.33**	**67.64**	42.64	57.27	72.60
DPR	39.16	42.53	65.27	31.11	45.16	67.06	44.17	57.48	73.03
DCCS	**43.62**	45.78	66.98	33.66	47.58	67.57	45.46	**58.89**	**74.28**

Table A2. Comparative results of data-to-text generation using GPT-3.5 on the ToTTo (100 test samples) in a 10-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

Method	Overall		Overlap Subset		Nonoverlap Subset	
	BLEU	PARENT	BLEU	PARENT	BLEU	PARENT
w/o example	26.2	50.1	28.8	50.7	23.9	49.5
Random	27.3	49.6	28.3	50.2	26.2	48.8
KATE	**38.2**	**58.1**	**42.8**	**62.3**	**32.7**	**52.8**
KFN	24.6	47.4	24.9	45.9	24.2	49.3
DPP	28.2	49.9	29.6	49.8	26.5	50.0
BM25	33.5	56.4	39.5	60.2	27.1	51.5
DPR	32.8	53.9	39.0	57.1	25.8	49.8
DCCS	35.2	55.2	39.0	57.7	30.8	52.0

Table A3. Comparative results of data-to-text generation using GLM-3 setting on the E2E, DART, and WebNLG (100 test samples) in a 10-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	E2E			DART			WebNLG		
Method	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
Random	52.39	50.56	68.28	45.68	57.19	73.22	56.49	66.46	78.22
KATE	54.37	**52.03**	69.44	47.91	58.13	74.14	62.52	70.71	81.10
KFN	52.93	49.84	57.67	46.55	57.48	72.69	60.87	69.12	79.35
DPP	53.02	50.51	69.11	47.39	57.10	73.53	59.47	69.40	79.83
BM25	53.36	50.64	64.15	46.99	58.97	73.79	59.56	69.11	79.51
DPR	54.99	50.60	68.90	47.84	57.95	74.33	60.17	67.96	79.42
DCCS	**55.15**	51.43	**69.81**	**49.49**	**59.29**	**75.22**	**63.99**	**71.17**	**81.70**

Table A4. Comparative results of data-to-text generation using Llama-3.1 on the E2E, DART, and WebNLG in a 10-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

	E2E			DART			WebNLG		
Method	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore	BLEU	ROUGE-L	BERTScore
Random	45.92	47.29	64.32	35.65	50.05	66.24	46.18	57.89	71.24
KATE	51.28	49.76	**68.75**	**37.78**	**51.21**	**67.24**	49.17	**59.74**	**73.01**
KFN	44.02	46.60	63.79	34.72	49.11	64.59	38.27	52.51	63.87
DPP	48.81	49.26	66.22	37.53	51.04	66.83	48.13	58.92	71.13
BM25	48.45	48.71	67.52	37.53	50.86	66.49	49.01	59.50	71.61
DPR	48.09	48.53	65.99	35.70	49.84	65.62	48.40	58.74	71.56
DCCS	**54.20**	**53.68**	68.73	37.59	51.12	66.84	**49.59**	59.51	72.86

Table A5. BLEU and PARENT on ToTTo using Llama-3.1 in a 10-shot setting. The best performance per metric is shown in **bold** and the second-best result is underlined

Method	Overall		Overlap subset		Nonoverlap subset	
	BLEU	PARENT	BLEU	PARENT	BLEU	PARENT
w/o example	24.3	48.18	28.1	49.99	21.1	46.39
Random	28.7	49.05	33.1	51.15	25.0	46.98
KATE	**42.0**	57.09	**52.0**	**61.94**	33.2	52.32
KFN	27.9	46.46	31.9	48.59	24.5	44.36
DPP	30.8	48.21	42.3	55.37	21.8	41.15
BM25	40.5	55.89	50.0	60.15	32.1	51.71
DPR	26.6	44.93	34.6	51.74	19.6	38.23
DCCS	41.9	**57.41**	51.6	61.80	**33.3**	**53.08**

Appendix B. Additional Batched Generation Experiment Results

Table B1. BLEU for batched generation on the E2E, WebNLG, DART and ToTTo using GPT-3.5 in a 10-shot setting. The best performance per dataset is shown in **bold**

Method	Batch size	E2E	DART	WebNLG	ToTTo
Random-batch	5	43.3	29.9	28.3	16.5
	10	33.8	31.6	25.7	19.2
Data-based centroid	5	46.7	36.4	43.9	26.7
	10	45.0	36.3	44.3	24.0
Text-based centroid	5	50.0	35.7	44.0	24.8
	10	48.0	34.3	44.8	25.1
DCCS-batch	5	**50.2**	**37.0**	**45.4**	**30.7**
	10	**48.5**	**36.9**	**45.3**	**27.1**

Table B2. BLEU for batched generation on the E2E, WebNLG, DART and ToTTo using GLM-3 in a 10-shot setting

Method	Batch size	E2E	DART	WebNLG	ToTTo
Random-batch	10	50.9	37.3	45.1	24.8
DCCS-batch	10	**53.1**	**38.3**	**51.3**	**29.1**

Table B3. BLEU for batched generation on the E2E, DART, WebNLG and ToTTo using Llama-3.1 in a 10-shot setting. The best performance per dataset is shown in **bold**

Method	Batch Size	E2E	DART	WebNLG	ToTTo
Random-batch	5	43.7	34.4	42.9	25.8
	10	44.2	34.5	42.8	26.8
Data-based centroid	5	47.8	35.1	43.7	25.5
	10	46.5	35.5	43.4	28.7
Text-based centroid	5	47.7	34.9	43.4	26.3
	10	46.7	35.3	43.4	28.7
DCCS-batch	5	**51.2**	**35.7**	**45.5**	**29.6**
	10	**49.1**	**36.8**	**46.5**	**30.3**

Appendix C. Additional Statistical Analysis Result

Table C1. Tukey–HSD pairwise comparisons on BLEU (Δ = G2–G1). Bold $p < 0.05$

E2E				DART			
Group 1	Group 2	Δ	p	Group 1	Group 2	Δ	p
DCCS	KATE	−0.030	**0.015**	DCCS	KATE	+0.001	0.778
DCCS	BM25	−0.058	**0.002**	DCCS	BM25	−0.006	0.201
DCCS	Random	−0.083	**0.001**	DCCS	Random	−0.022	**0.042**
KATE	BM25	−0.028	**0.018**	KATE	BM25	−0.007	0.178
KATE	Random	−0.053	**0.003**	KATE	Random	−0.023	**0.039**
BM25	Random	−0.025	**0.022**	BM25	Random	−0.016	0.074
WebNLG				ToTTo			
Group 1	Group 2	Δ	p	Group 1	Group 2	Δ	p
DCCS	KATE	−0.004	0.223	DCCS	KATE	−0.001	0.905
DCCS	BM25	−0.007	0.153	DCCS	BM25	−0.020	**0.048**
DCCS	Random	−0.035	**0.006**	DCCS	Random	−0.133	**0.000**
KATE	BM25	−0.003	0.314	KATE	BM25	−0.019	0.052
KATE	Random	−0.031	**0.011**	KATE	Random	−0.132	**0.000**
BM25	Random	−0.028	**0.018**	BM25	Random	−0.113	**0.000**

Appendix D. Choice of the Secondary-Cluster Size

Following prior work on in-context example selection Liu *et al.* (2022b), we restrict the number of secondary clusters – and thus the number of in-context examples – to $m \in \{5, 10\}$ throughout the paper. Figure D1 illustrates how BLEU varies with the number of in-context examples on the three datasets. All results were obtained with LLaMA-3.1-8B, using 100 randomly sampled test instances from each dataset. Preliminary runs showed that using fewer than five examples reduced performance, whereas increasing m beyond ten yielded no consistent BLEU gain while incurring noticeably higher prompt cost and latency. We therefore adopt $m = 5$ as the default setting and $m = 10$ as a high-accuracy variant.

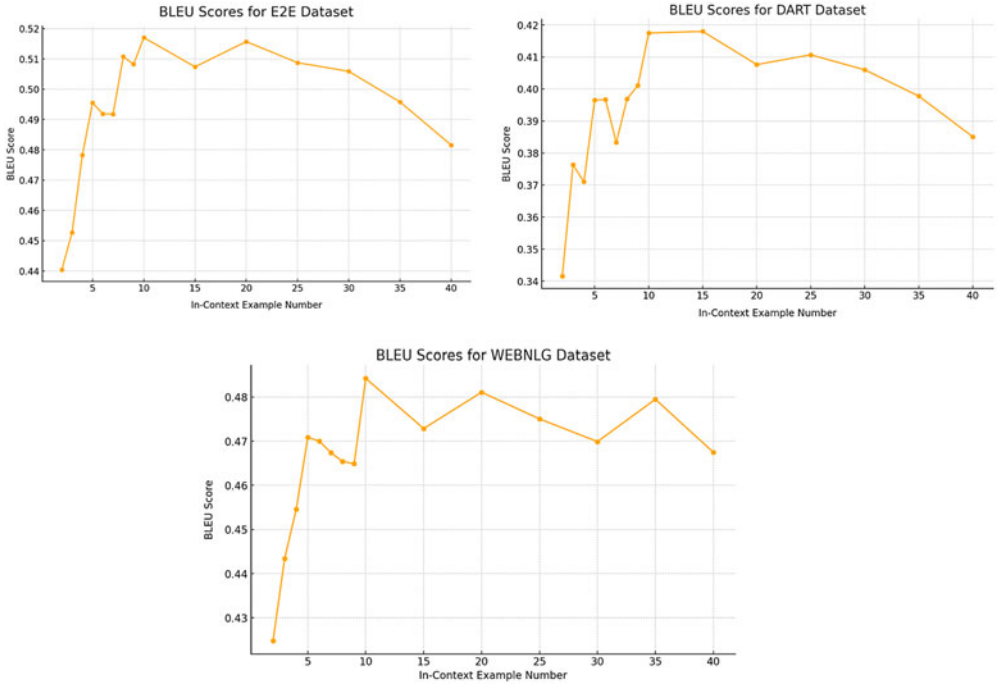

Figure D1. BLEU scores on the E2E, DART, and WebNLG as a function of the number of in-context examples (m).

Cite this article: Li Y, Yang J, Jiang L, Liu S and An N. How to quickly select good in-context examples in large language models for data-to-text tasks?. *Natural Language Processing* https://doi.org/10.1017/nlp.2025.10010