# UCS: Estimating Unseen Coverage for Improved In-Context Learning

**Jiayi Xin[1], Xiang Li[1], Evan Qiang[1], Weiqing He[1], Tianqi Shang[1], Weijie J. Su[1], Qi Long[1]**

[1]University of Pennsylvania

**Correspondence:** jiayixin@seas.upenn.edu, suw@wharton.upenn.edu and qlong@upenn.edu

## Abstract

In-context learning (ICL) performance depends critically on which demonstrations are placed in the prompt, yet most existing selectors prioritize heuristic notions of relevance or diversity and provide limited insight into the *coverage* of a demonstration set. We propose **Unseen Coverage Selection (UCS)**, a training-free, subset-level coverage prior motivated by the principle that a good demonstration set should *expose the model to latent cluster unrevealed by the currently selected subset*. UCS operationalizes this idea by **(1)** inducing discrete latent *clusters* from model-consistent embeddings and **(2)** estimating the number of unrevealed clusters within a candidate subset via a Smoothed Good–Turing estimator from its empirical frequency spectrum. Unlike previous selection methods, UCS is coverage-based and training-free, and can be seamlessly combined with both query-dependent and query-independent selection baselines via a simple regularized objective. Experiments on multiple intent-classification and reasoning benchmarks with frontier LLMs show that augmenting strong baselines with UCS consistently improves ICL accuracy by up to **2–6%** under the same selection budget, while also yielding insights into task- and model-level latent cluster distributions. Code is available at `https://github.com/Raina-Xin/UCS`.

## 1 Introduction

In-context learning (ICL) enables large language models (LLMs) to perform new tasks by conditioning on a small set of input–output demonstrations, without parameter updates (Brown et al., 2020). While highly effective across tasks (Wei et al., 2022a; Wang et al., 2024), ICL is notoriously sensitive to the choice of demonstrations—under the same selection budget, different demonstration sets can lead to large performance variations (Su et al., 2022; Min et al., 2022b). This sensitivity poses a central challenge in modern deployments, where ICL demonstrations are selected from large candidate pools under limited annotation budgets (Wang et al., 2025). Although a large body of work on automatic demonstration selection has improved ICL performance—via similarity-based (Ye et al., 2023a; Su et al., 2022), diversity-based (Luo et al., 2023; Yang et al., 2023), and information-theoretic methods (Wu et al., 2023b; Van et al., 2024; Qin et al., 2024; Scarlatos and Lan, 2023; Wang et al., 2025)—it remains unclear how much of the task's underlying *latent cluster space* is revealed by the chosen subset.

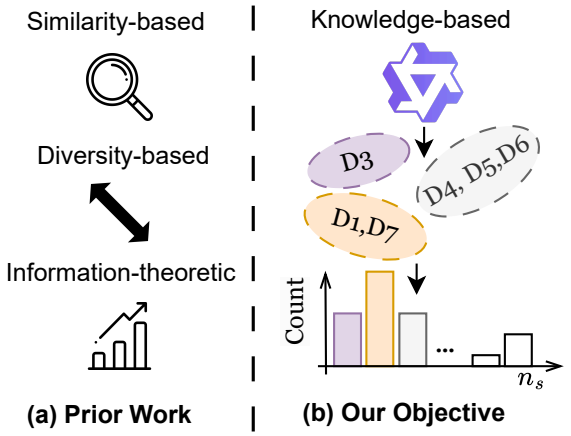


Figure 1: Contrast to prior methods (**left**), our approach reasons at the subset level by clustering demonstrations into latent clusters and using their frequency to measure coverage (**right**).

In this work, we argue that *coverage* provides a complementary dimension for reasoning about demonstration selection (Figure 1). Prior studies suggest that ICL performance depends not only on relevance to the test input, but also on exposure to diverse latent concepts and skills (Min et al., 2022b; Peng et al., 2024). From our perspective, a demonstration is valuable precisely when it introduces *previously unseen latent clusters*. Nevertheless, latent cluster coverage alone is insufficient to serve as a



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 10965–10981

standalone criterion, since relevance- and diversity-based methods capture important instance-level signals. We therefore treat coverage as a *prior* that biases demonstration selection toward subsets with broader latent cluster exposure, rather than as a replacement for existing objectives.

To operationalize this idea, we introduce **Unseen Coverage Selection (UCS)**, a statistically principled framework for estimating latent cluster coverage in ICL demonstration selection. At a high level, UCS entails two steps: it first clusters model embeddings of demonstrations to induce a set of discrete latent *clusters*, and then applies the Smoothed Good–Turing estimator (Orlitsky et al., 2016) to the resulting empirical frequency spectrum of a candidate subset to estimate how many additional clusters are likely to remain *unrevealed*—that is, latent clusters not yet covered by the selected demonstration subset under the same candidate-pool sampling process. This estimate provides a subset-level measure of cluster coverage, which we use as a lightweight coverage prior that can be incorporated into existing selection objectives. As a result, UCS is training-free, and it can be layered on top of both query-dependent and query-independent methods via a simple regularized objective with minimal computational overhead.

We evaluate UCS across three intent classification benchmarks, three Big-Bench Extra Hard (BBEH) reasoning tasks, and three frontier LLMs. Our experiments show that augmenting strong baseline selectors with UCS consistently improves ICL accuracy by up to **2–6%** under the same selection budget. Beyond performance gains, UCS provides interpretable insights into how latent clusters are distributed across tasks and models, revealing systematic differences in the prevalence of common versus rare clusters.

Our contributions are summarized as follows:

- ⋆ We introduce *coverage* as a complementary subset-level objective for ICL demonstration selection, focusing on estimating unseen latent clusters beyond instance-level relevance.
- ⋆ We propose UCS, a lightweight, training-free coverage prior based on the Smoothed Good–Turing estimation that can be plugged into existing selection methods.
- ⋆ UCS consistently improves ICL accuracy across three intent classification datasets, three BBEH reasoning tasks, and three LLM backbones, achieving gains of up to **6.2%** for query-independent selection and stable improvements for strong query-dependent baselines under the same budget.

## 2 Related Work

**In-Context Learning and Demonstration Selection.** ICL performance is highly sensitive to the choice of demonstrations: under a fixed budget, different selections can lead to large accuracy variations (Su et al., 2022; Min et al., 2022b). Early work addresses this problem using heuristic similarity or diversity criteria, selecting demonstrations that are either semantically close to the test query or diverse among themselves (Liu et al., 2022; Lu et al., 2022; Wu et al., 2023b; Su et al., 2022; Ye et al., 2023a; Levy et al., 2023; Luo et al., 2023; Agrawal et al., 2023). The OpenICL framework (Wu et al., 2023a) provides a unified evaluation of such strategies and includes representative methods based on similarity (e.g., VoteK) (Su et al., 2022), diversity via determinantal point processes (DPP) (Yang et al., 2023), and information-theoretic criteria such as minimum description length (MDL) (Wu et al., 2023b). However, these approaches do not explicitly characterize how much of the task's underlying *latent space* is exposed by a selected demonstration subset.

**Query-Adaptive and Iterative Retrieval.** Several methods make demonstration selection query-adaptive by incorporating relevance signals, iterative refinement, or learned policies. Influence-based approaches such as InfICL estimate the effect of individual training examples on predictions (Van et al., 2024), while iterative methods (e.g., IDS) refine demonstrations using intermediate chain-of-thought signals (Qin et al., 2024). These ideas are closely related to reasoning-based prompting techniques, including chain-of-thought, zero-shot reasoning, self-consistency, and automatic CoT construction (Wei et al., 2022b; Kojima et al., 2022; Wang et al., 2022; Zhang et al., 2022). Other work formulates selection as a sequential decision problem, optimized via reinforcement learning or query-conditioned mechanisms (Scarlatos and Lan, 2023; Wang et al., 2025; Liu et al., 2022; An et al., 2023; Liu et al., 2024), with recent extensions exploring alternative reward designs (Zhang et al., 2025). These methods emphasize instance-level optimization but typically require additional training or incur non-trivial inference overhead.

**Subset-Level Coverage and Unseen Information.** Beyond instance-level and query-adaptive strategies, prior work has examined demonstration selection from a *subset-level* perspective, emphasizing latent coverage rather than relevance alone. Empirical studies show that ICL benefits from exposing diverse concepts and skills, even when they are not semantically closest to the test query (Min et al., 2022b; Peng et al., 2024; Levy et al., 2023; Gupta et al., 2023; Ye et al., 2023b; An et al., 2023; Lu et al., 2022; Min et al., 2022a). This view aligns with diversity- and representativeness-based objectives, such as DPP and related subset selection methods, which trade off redundancy and coverage under a fixed budget (Yang et al., 2023; Kulesza et al., 2012; Lin and Bilmes, 2011). However, most existing approaches lack a principled way to quantify how much latent structure is covered or remains unseen in a selected set (Wu et al., 2023b; Qin et al., 2024). This limitation is closely related to classical unseen-species estimation (Orlitsky et al., 2016) and recent efforts to quantify unseen knowledge in LLM evaluation (e.g., KnowSum) (Li et al., 2025), which do not address ICL demonstration selection under a fixed budget.

# 3 Unseen Coverage Selection for In-Context Learning

## 3.1 Preliminaries and Notation

**Problem setup.**

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ denote a pool of labeled demonstrations, and let $B$ denote the selection budget. For each test input $x_{\text{test}}$, the goal is to select a subset $S \subseteq \mathcal{D}$ with $|S| = B$ and construct an in-context prompt by concatenating the labeled examples in $S$, followed by $x_{\text{test}}$.

**Base demonstration selectors.** We view existing ICL demonstration selection methods as defining a subset utility function $U_{\text{base}}(S; x_{\text{test}})$. Some selectors are *query-dependent*, which produce different subset for each test input (e.g., DPP and MDL), while others are *query-independent* that select a single global subset shared across all test examples (e.g., VoteK).

## 3.2 Algorithm Overview of UCS

Our key hypothesis is that strong demonstration sets should *cover* a broad range of latent units in the pool, rather than over-sampling redundant regions. To operationalize this idea, we introduce UCS (Unseen Coverage Selection) as a *subset-level coverage prior* that can be *layered on top of* existing demonstration selection methods. UCS is *not* used as a standalone retriever; instead, it regularizes the selection objective by biasing it toward subsets with higher expected coverage under a fixed selection budget.

Concretely, UCS consists of three stages: (i) representing demonstrations in a model-consistent embedding space (Section 3.3); (ii) inducing discrete latent clusters via dictionary learning and clustering (Section 3.4); and (iii) estimating subset-level coverage using a Smoothed Good–Turing estimator (Section 3.5). These components are combined with existing ICL selectors through a UCS-regularized objective, which we instantiate for DPP, MDL, and VoteK in Section 3.6. Figure 2 illustrates the end-to-end workflow.

## 3.3 Model-Consistent Representation

UCS is designed to be *model-consistent*: the representation space used to define latent clusters should align with the LLM used at inference time for ICL. Accordingly, we embed the demonstration pool using the *same* backbone model during evaluation, so that distances and clusters reflect the inductive biases of the model being prompted.

**Embedding procedure.** For each demonstration $(x_i, y_i) \in \mathcal{D}$, we embed *only the input* $x_i$ (excluding labels) to avoid leaking task-specific surface information into the representation. Let $\mathbf{H}_i^{(\ell)} \in \mathbb{R}^{T_i \times d}$ denote the hidden states at layer $\ell$ for the tokenized input $x_i$, with an attention mask $\mathbf{m}_i \in \{0,1\}^{T_i}$ indicating non-padding tokens. We obtain a fixed-length embedding $\mathbf{e}_i \in \mathbb{R}^d$ via **masked mean pooling**:

$$\mathbf{e}_i = \frac{\sum_{t=1}^{T_i} m_{i,t}\, \mathbf{H}_{i,t}^{(\ell)}}{\sum_{t=1}^{T_i} m_{i,t}}. \tag{1}$$

The resulting $\{\mathbf{e}_i\}_{i=1}^{N}$ serve as the continuous representation of the demonstration pool for subsequent clustering and coverage estimation. Additional choices, including the layer $\ell$, alternative pooling strategies, dimensionality reduction, are deferred to Appendix A.1.

## 3.4 Inducing Discrete Clusters from Continuous Embeddings

Smoothed Good–Turing–style unseen estimators operate on *discrete objects*. To apply them to continuous LLM embeddings, we discretize each embedding into a *cluster assignment* via a three-step

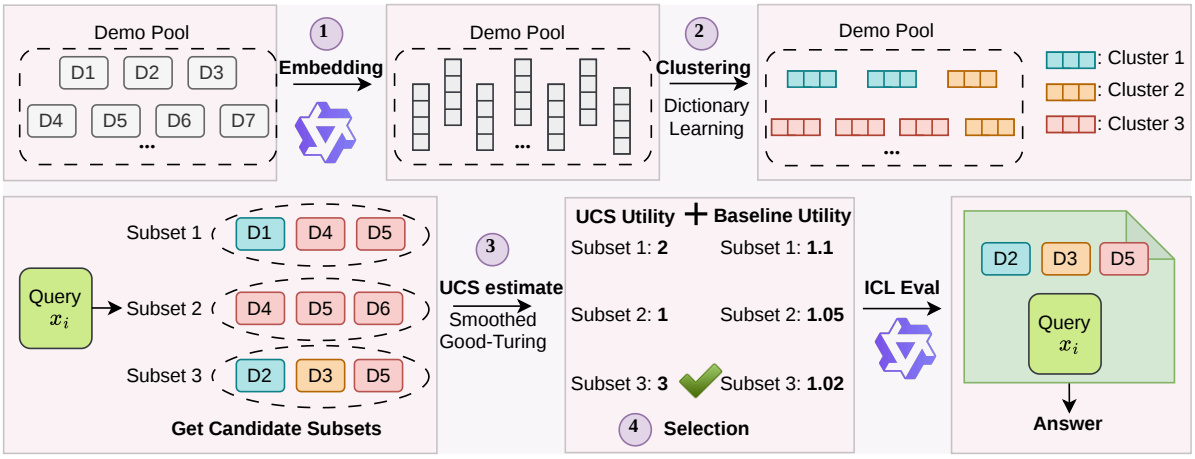


Figure 2: Schematics of UCS. **(1)** Demonstrations are embedded in a model-consistent space. **(2)** Embeddings are discretized into latent clusters via dictionary learning and clustering. **(3)** Candidate subsets are scored using an SGT estimation-regularized objective with a base selection utility. **(4)** The selected subset is used for ICL.

procedure: (i) learning a latent dictionary over embeddings, (ii) encoding each example as a structured code over dictionary atoms, and (iii) clustering these codes to induce discrete units.

**Dictionary learning as a latent basis.** Let $E = [e_1; \ldots; e_N] \in \mathbb{R}^{N \times d}$ denote the embedding matrix of the demonstration pool. For numerical stability and computational efficiency, we optionally standardize features and apply PCA to obtain $\tilde{E} \in \mathbb{R}^{N \times d'}$, where $d' \ll d$ in our implementation. We then learn a dictionary with $K$ atoms by approximating each embedding as

$$\tilde{e}_i \approx D r_i, \qquad r_i \in \mathbb{R}^K, \tag{2}$$

where $D \in \mathbb{R}^{d' \times K}$ is the learned basis and $r_i$ is the code for example $i$. We adopt **ridge coding** to avoid pathological sparsity and to preserve meaningful geometry in the code space. After fitting $D$, we compute codes via the ridge-regularized projection

$$r_i = \arg\min_{r \in \mathbb{R}^K} \|\tilde{e}_i - Dr\|_2^2 + \alpha \|r\|_2^2, \tag{3}$$

which admits a closed-form solution and can be efficiently applied in batch.

**Clustering in code space.** A naive discretization strategy would assign each example to its highest-magnitude atom (i.e., $\arg\max_j |r_{ij}|$), but this ignores the inherently *compositional* structure of the codes. Instead, we treat the full code vector as a latent *cluster signature* and perform clustering directly in the code space. Because code norms can vary due to input length and embedding scale, we normalize each code before clustering:

$$\bar{r}_i = \frac{r_i}{\|r_i\|_2 + \varepsilon}. \tag{4}$$

We then apply DBSCAN (Ester et al., 1996) to $\{\bar{r}_i\}_{i=1}^N$ using cosine distance, producing cluster assignments $c_i \in \{-1, 0, 1, 2, \ldots\}$. Points labeled as noise ($c_i = -1$) are each assigned a unique singleton cluster. After this remapping, we obtain $c_i' \in \{1, 2, \ldots\}$, which serves as the **discrete cluster ID** required by the SGT estimator; each cluster serves as a discrete latent type for downstream coverage estimation.

**Why dictionary learning plus clustering?** Empirically, atom activations exhibit a heavy-tailed distribution: a small number of atoms are frequently used, while many others are rare. As a result, argmax-based assignment over-emphasizes frequent atoms and fails to capture multi-atom structures. In contrast, clustering in the code space groups examples that share similar *combinations* of atoms, yielding clusters that better reflect recurring latent patterns while preserving a long tail of fine-grained units. Additional discussion and implementation details, including the DBSCAN `eps` heuristic and comparisons with argmax discretization, are provided in Appendix A.2 and Appendix A.3.

### 3.5 UCS Coverage Functional

**Frequency spectrum of clusters.** Let $S$ be a candidate demonstration subset and let $c_i$ denote the discrete cluster assigned to example $i$ (Section 3.4).

We define the multiplicity of unit $u$ in $S$ by

$$n_u(S) = \sum_{i \in S} \mathbb{I}[c_i = u], \tag{5}$$

and the corresponding frequency-of-frequencies (or *spectrum*) by

$$f_s(S) = \big|\{u : n_u(S) = s\}\big|. \tag{6}$$

**Smoothed Good–Turing estimation.** Given the frequency spectrum $\{f_s(S)\}_{s \geq 1}$, we apply a Smoothed Good–Turing (SGT) estimator (Good, 1953; Orlitsky et al., 2016) to predict the number of *new* clusters that would be observed if we were to draw $m$ additional samples from the same candidate pool (we define the expansion factor $t := m/|S|$). Let $\widehat{U}_t(S)$ denote this unseen estimate. The classical Good–Turing functional is

$$\widehat{U}_t^{\mathrm{GT}}(S) = -\sum_{s=1}^{M} (-t)^s \, f_s(S), \tag{7}$$

where we truncate the alternating series at a maximum count $M$ (default $M$=20) to reduce variance from sparse high-multiplicity bins. To further stabilize extrapolation, we apply offset-damped weights $w_s(t, \alpha) \in [0, 1]$ that discount higher-order terms:

$$\widehat{U}_{t,M}^{\mathrm{SGT}}(S) = -\sum_{s=1}^{M} (-t)^s \, w_s(t, \alpha) \, f_s(S), \tag{8}$$

with negative or non-finite values clamped to zero. Full details on the weighting scheme and hyperparameters are in Appendix A.4.

**UCS coverage functional.** Instead of optimizing the unseen mass alone, we define the UCS coverage functional $\Phi_{\mathrm{UCS}}(S)$ as the estimated total number of clusters, i.e., $\widehat{K}_t(S)$, in a set $S$:

$$\Phi_{\mathrm{UCS}}(S) := \widehat{K}_t(S) = K_{\mathrm{seen}}(S) + \widehat{U}_t(S), \tag{9}$$

$$K_{\mathrm{seen}}(S) = \big|\{u : n_u(S) > 0\}\big|. \tag{10}$$

Maximizing $\Phi_{\mathrm{UCS}}(S)$ encourages demonstration subsets that both cover many distinct clusters and exhibit frequency spectra indicative of additional unseen clusters. For numerical robustness, negative or non-finite values of $\widehat{U}_t(S)$ are clamped to zero, and designated noise labels (e.g., DBSCAN noise) are excluded from the frequency spectrum.

| Methods | UCS-augmented scoring rule |
|---|---|
| DPP+UCS | $\arg\max_{i \notin S} \ \Delta_{\mathrm{DPP}}(i \mid S) + \lambda(\Phi(S \cup \{i\}) - \Phi(S))$ |
| MDL+UCS | $\arg\max_{S \in \mathcal{C}(x)} \ \mathrm{MDL}(S; x) + \lambda \Phi(S)$ |
| VoteK+UCS | $\mathrm{score}(i) = v(i) + \lambda \log w_{c(i)}, \ \ w_c \propto (\widehat{g}_{n_c} + \epsilon)^{-1}$ |

Table 1: Instantiations of UCS for three baselines.

### 3.6 UCS-Regularized Selection

UCS is not a standalone demonstration selector. Instead, it defines a *set-level coverage prior* that regularizes existing in-context learning selection objectives. Given a budget $B$, we select demonstrations by solving

$$S^* = \arg\max_{|S|=B} \Big( U_{\mathrm{base}}(S; x_{\mathrm{test}}) + \lambda \, \Phi_{\mathrm{UCS}}(S) \Big), \tag{11}$$

where $U_{\mathrm{base}}$ denotes the utility induced by a base selector (e.g., DPP, MDL, or VoteK), $\Phi_{\mathrm{UCS}}(S)$ is the UCS coverage functional defined in Section 3.5, and $\lambda \geq 0$ controls the strength of the coverage regularization.

Crucially, $\Phi_{\mathrm{UCS}}$ operates at the *subset level*: it depends on the joint frequency spectrum induced by $S$ and does not decompose into per-example scores. As a result, it can be interpreted as an *unnormalized prior* over subsets, biasing selection toward demonstration sets with broader and more extensible coverage. Throughout this work, UCS is always used in conjunction with a base selector and never optimized in isolation.

**Instantiations with existing selectors.** Table 1 summarizes how the UCS coverage functional is incorporated into three standard demonstration selectors. Across all cases, the UCS functional is added as a lightweight regularization term on top of the original objective, without modifying the underlying retrieval pipeline. Setting $\lambda = 0$ recovers the original formulations. More implementation details are deferred to Appendix C.

## 4 Experiment Results

### 4.1 Setup

We evaluate UCS on three intent classification benchmarks: BANKING77 (Casanueva et al., 2020) (banking-domain intents, $n$ = 10,003), CLINC150 (Larson et al., 2019) (large-scale multi-domain intents, $n$ = 15,000), and HWU64 (Liu et al., 2019) (fine-grained conversational intents, $n$ = 11,036). To test transfer beyond intent classification, we further evaluate on three Big-Bench Extra Hard (BBEH) reasoning tasks (Kazemi et al.,

2025): *Causal Understanding*, *Object Properties*, and *Shuffled Objects*, each containing 200 examples split 80/20 into candidate pool and evaluation set. For each dataset, demonstrations are selected from the training split, and unless otherwise specified, we fix the selection budget to $B$=10 examples per query.

To assess robustness across model scales and families, we consider three frontier LLMs: `Qwen2.5-7B-Instruct` (Yang et al., 2024), `Llama-3.2-3B-Instruct` (Dubey et al., 2024), and `Gemma-2-9B-it` (Team et al., 2024). We consider three representative demonstration selectors: VOTEK, a query-independent baseline, and DPP and MDL, which are query-dependent; UCS is implemented as a plug-in prior on top of each selector, as summarized in Table 1, using the OpenICL (Wu et al., 2023a) codebase[1].

### 4.2 UCS Improves ICL Demonstration Selection

Table 2 summarizes ICL accuracy across three intent classification benchmarks and three backbone models. Overall, augmenting existing selectors with UCS consistently improves or matches performance, with the largest gains observed for query-independent selection. ❶ **Significant gains for query-independent selection.** UCS substantially improves VOTEK, particularly on challenging tasks. On HWU64, UCS+VoteK yields accuracy gains of **+6.2%** (Qwen2.5-7B) and **+4.1%** (Llama-3.2-3B). Similarly, on CLINC150, it boosts Qwen2.5-7B performance by **+4.1%**. ❷ **Complementary to query-dependent selectors.** When combined with MDL or DPP, UCS yields modest but consistent improvements in several settings (e.g., **+1.4%** on BANKING77 for Llama-3.2-3B), while preserving performance elsewhere. ❸ **Robust across models and datasets.** Across nine dataset–LLM pairs, UCS improves or matches the base selector in most cases, with larger relative gains in lower-accuracy regimes. ▷ **Takeaway.** UCS is a lightweight, model-agnostic plug-in that strengthens ICL demonstration selection by explicitly encouraging coverage of unrevealed clusters. A detailed runtime breakdown is provided in Appendix B: offline preprocessing takes 38–57 s per dataset, and for our recommended pairings (MDL+UCS, VoteK+UCS), the additional online overhead is typically ∼0–3 s.

Table 3 presents results on three BBEH reasoning tasks. On Qwen2.5-7B, UCS improves 8 of 9 selector–task settings, with gains up to **+12.5 pp** on Shuffled Objects (DPP→UCS+DPP) and **+8.4 pp** on Causal Understanding (MDL→UCS+MDL), suggesting strong transfer to reasoning tasks.

### 4.3 UCS Reveals Cluster ID Distribution Across Tasks and LLMs

To characterize how cluster ID is distributed in the demonstration pool, we analyze the cluster statistics induced by unsupervised clustering. Figure 3 visualizes the cluster-size distribution for each dataset–LLM pair, including the mass of small clusters (left) and the sizes of the largest clusters (right). Across all settings, the distributions are highly skewed: most demonstrations belong to small or singleton clusters, while a small number of clusters dominate the pool. This long-tailed pattern indicates that a few cluster types are frequently observed, whereas many others appear rarely. Importantly, both the degree of skew and the dominant clusters vary substantially across datasets and LLMs. ▷ **Takeaway.** Cluster ID distributions vary substantially across tasks and LLMs, motivating UCS to explicitly account for uneven and model-specific coverage.

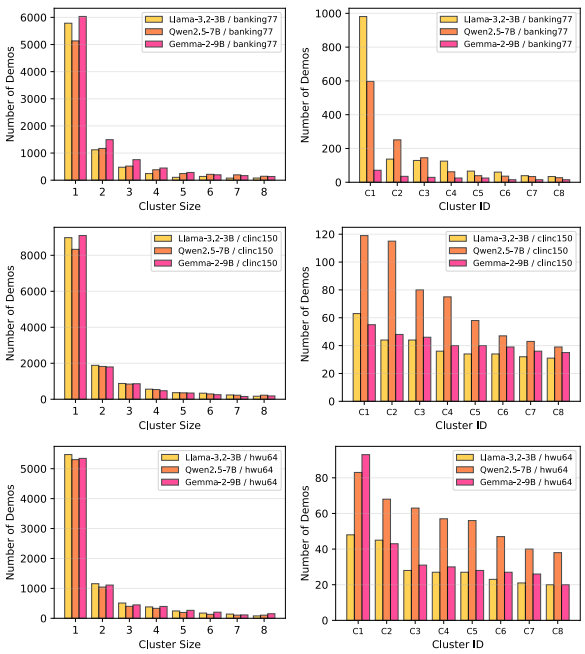


Figure 3: **Cluster-size distributions across datasets and LLMs.** Left: number of demonstrations in clusters of size $k \in [1, 8]$ (heavy-tailed, many singletons). Right: sizes of the top-8 clusters (a few dominant clusters).

[1]Our implementation is available at https://github.com/Raina-Xin/UCS/tree/main/OpenICL/openicl/icl_retriever

Table 2: **UCS consistently improves base selection across different datasets and models.** ICL accuracy (higher is better) across three intent classification datasets. Each method selects a demonstration set of size **budget=10** using `dict_dbscan` clustering. We evaluate on **test_size=500** examples and report mean±std over **n_runs=3**.

| Backbone Model | Dataset | MDL | UCS+MDL | DPP | UCS+DPP | VoteK | UCS+VoteK |
|---|---|---|---|---|---|---|---|
| Qwen2.5-7B-it | Banking77 | 0.764±0.019 | **0.771**±0.011 | **0.831**±0.005 | **0.831**±0.007 | 0.518±0.008 | **0.543**±0.020 |
| | CLINC150 | 0.748±0.004 | **0.752**±0.012 | 0.755±0.011 | **0.775**±0.010 | 0.703±0.001 | **0.744**±0.010 |
| | HWU64 | 0.785±0.002 | **0.801**±0.011 | 0.791±0.003 | **0.794**±0.004 | 0.609±0.003 | **0.671**±0.021 |
| Llama-3.2-3B-it | Banking77 | 0.718±0.003 | **0.732**±0.012 | 0.743±0.008 | **0.746**±0.006 | 0.421±0.009 | **0.426**±0.002 |
| | CLINC150 | 0.747±0.005 | **0.755**±0.006 | 0.698±0.008 | **0.714**±0.009 | 0.597±0.003 | **0.599**±0.008 |
| | HWU64 | 0.537±0.008 | **0.539**±0.010 | 0.565±0.003 | **0.571**±0.008 | 0.457±0.004 | **0.498**±0.020 |
| Gemma-2-9B-it | Banking77 | 0.883±0.007 | **0.893**±0.007 | 0.915±0.001 | **0.916**±0.001 | 0.665±0.002 | **0.677**±0.004 |
| | CLINC150 | 0.977±0.005 | **0.981**±0.006 | **0.990**±0.000 | **0.990**±0.000 | 0.849±0.001 | **0.852**±0.004 |
| | HWU64 | 0.887±0.005 | **0.892**±0.005 | 0.906±0.002 | **0.907**±0.002 | **0.794**±0.002 | **0.794**±0.002 |

Table 3: **UCS consistently improves base selection on reasoning tasks (Qwen2.5-7B).** ICL accuracy (higher is better) across three Big-Bench Extra Hard (BBEH) reasoning datasets (Kazemi et al., 2025): Causal Understanding, Object Properties, and Shuffled Objects. Each method selects a demonstration set of size **budget=10** using `dict_dbscan` clustering. We evaluate on **test_size=40** examples and report mean±std over **n_runs=3**.

| Backbone Model | Dataset | MDL | UCS+MDL | DPP | UCS+DPP | VoteK | UCS+VoteK |
|---|---|---|---|---|---|---|---|
| Qwen2.5-7B-it | Causal | 0.608±0.012 | **0.692**±0.042 | 0.517±0.012 | **0.542**±0.012 | 0.492±0.012 | **0.525**±0.00 |
| | Properties | 0.083±0.029 | **0.092**±0.031 | 0.050±0.020 | **0.067**±0.024 | 0.050±0.020 | **0.067**±0.012 |
| | Shuffled | 0.200±0.043 | **0.225**±0.000 | 0.133±0.012 | **0.258**±0.012 | 0.242±0.063 | **0.242**±0.043 |

### 4.4 UCS Prioritizes Cluster Coverage

To quantify how UCS alters the content of the prompts, we analyze the latent cluster statistics of selected subsets in Table 4. Driven by the SGT-based coverage functional, UCS explicitly biases selection toward the "rare" clusters located in the heavy tail of the distribution (as identified in Figure 3). This is empirically confirmed by the shift in cluster composition of retrieved demonstrations: UCS-augmented subsets generally exhibit a higher count of unique clusters and a smaller average cluster size compared to their baselines. This effect is most pronounced for VoteK, where adding UCS eliminates redundancy entirely, reducing the average cluster size from 8.50 to 1.00. By targeting these rare clusters, our method effectively exchanges marginal redundancy for broader exposure to unseen cluster types. ▷ **Takeaway.** UCS diversifies prompt content by prioritizing the heavy tail of the cluster distribution, ensuring that selected demonstrations maximize coverage of distinct semantic concepts.

**Qualitative audit on BANKING77.** To assess whether the induced clusters are semantically meaningful, we examine representative clusters on BANKING77 (Qwen2.5-7B). Many clusters correspond to coherent, human-recognizable micro-topics: e.g., cluster 380 (size=7) is entirely *unable_to_verify_identity* paraphrases, cluster 298 (size=5) concentrates on *physical card vs. virtual card*, and cluster 562 (size=4) is fully *transaction_charged_twice*. When adding UCS on top of DPP, selections shift away from an over-broad dominant cluster (cid=0; size=8,970) toward smaller, semantically sharper clusters (e.g., cid=380, identity verification; cid=232, ATM cash issues). This behavior reflects UCS's frequency-spectrum mechanism: it penalizes over-representation of dominant modes and increases coverage of underrepresented latent clusters.

## 5 In-depth Analysis of UCS

### 5.1 Joint-Dictionary Extension: Cross-Model Alignment

As an exploratory extension, we test whether aligning embeddings from multiple LLMs into a shared representation improves robustness. We extend UCS to a **Joint Dictionary Learning** framework. Rather than deriving clusters from a single latent space, we align embeddings from multiple LLMs into a shared canonical representation. Formally, given embedding matrices $E^{(m)} \in \mathbb{R}^{N \times d_m}$ from $M$ different models, we learn a shared dictionary

Table 4: **UCS Prioritizes Cluster Coverage.** Metrics for Qwen on CLINC150. **#Uniq Clusters**: number of distinct clusters in the selected prompt. **Cluster Size**: average size of the clusters that the selected demos belong to (over the entire pool). **1/Size**: per-demo inverse cluster size averaged over the prompt; a value of 1.0 indicates all demos come from singleton clusters.

| Method | # Uniq Clusters | Cluster Size | 1/Size |
|---|---|---|---|
| MDL | 9.53±0.006 | 6.28±0.550 | 0.689±0.007 |
| UCS+MDL | 9.49±0.031 | 6.57±0.734 | 0.688±0.005 |
| DPP | 9.62±0.014 | 6.29±0.323 | 0.701±0.003 |
| UCS+DPP | 9.93±0.016 | 5.39±0.178 | 0.719±0.004 |
| VoteK | 9.67±0.471 | 8.50±1.390 | 0.604±0.020 |
| UCS+VoteK | 10.0±0.000 | 1.00±0.000 | 1.00±0.000 |

$D$ and shared sparse codes $R$ by optimizing orthogonal transformation matrices $B^{(m)}$ for each model. This is achieved by solving a joint reconstruction objective subject to orthogonal constraints:

$$\min_{D,R,\{B^{(m)}\}} \sum_{m=1}^{M} \left\| E^{(m)} B^{(m)} - R D^{\top} \right\|_F^2, \quad \text{s.t. } (B^{(m)})^{\top} B^{(m)} = I. \tag{12}$$

By mapping disparate embedding spaces into this unified coordinate system, we aim to capture semantic invariants agreed upon by multiple models. We evaluate this approach by constructing UCS priors using a joint dictionary learned from Qwen2.5-7B, Llama-3.2-3B, and Gemma-2-9B. Table 5 presents the ICL accuracy comparisons. Contrary to the expectation that multi-model consensus yields more robust priors, the results indicate a performance trade-off. While the joint dictionary maintains parity for the smaller Llama-3.2-3B backbone (e.g., maintaining 74.6% on Banking77 and yielding a marginal gain on HWU64), it causes significant degradation for the stronger Qwen2.5-7B model (e.g., −11.9% on HWU64). This suggests that forcing disparate embedding manifolds into a shared linear subspace may dilute the fine-grained semantic distinctions present in higher-capacity models, highlighting the difficulty of lossless alignment across heterogeneous architectures. These results further motivate our default choice of *model-consistent* (single-source) embeddings.

### 5.2 Sensitivity Analysis of UCS

**Sensitivity to SGT hyperparameters.** We examine the robustness of UCS to the hyperparameters of the SGT estimator by varying the extrapolation horizon $t$ and the frequency bin size, while fixing the base retriever. As shown in Table 6, UCS consistently improves over the base retriever across a wide range of SGT settings. The paired improvements remain stable as $t$ and the bin size change, indicating that UCS is robust to different SGT hyperparameters.

Table 5: **Single-Source vs. Joint-Source UCS.** Comparison of **UCS+DPP** accuracy ($B$=10) when using a dictionary derived from the inference model itself (Single-Source) versus a Joint Dictionary aligned across multiple LLMs. The joint representation maintains performance for the 3B model but shows trade-offs for the larger 7B model.

| Backbone | Dataset | Single-Source | Joint-Source | Δ |
|---|---|---|---|---|
| Qwen | Banking77 | 83.1±0.7 | 78.3±1.3 | −4.8 |
| | CLINC150 | 77.5±0.1 | 82.3±2.0 | +4.8 |
| | HWU64 | 79.4±0.4 | 67.5±1.3 | −11.9 |
| Llama | Banking77 | 74.6±0.6 | 74.6±1.1 | 0.0 |
| | CLINC150 | 71.4±0.9 | 70.5±1.3 | −0.9 |
| | HWU64 | 57.1±0.8 | 57.7±4.7 | +0.6 |

Table 6: **Sensitivity to SGT hyperparameters.** Accuracy and improvement for MDL and UCS+MDL for Llama model on Banking77 dataset.

| $t$ | bin size | UCS+MDL | Δ |
|---|---|---|---|
| 2 | 10 | 73.1±1.1 | +1.3 |
| 4 | 10 | 73.0±0.7 | +1.2 |
| 8 | 10 | 72.9±1.6 | +1.1 |
| 4 | 5 | 72.8±1.7 | +1.0 |
| 4 | 20 | 72.9±1.2 | +1.1 |

**Sensitivity to selection budget.** We vary the number of in-context demonstrations to examine whether UCS remains effective across different selection budget. As shown in Table 7, UCS consistently improves the base retriever across a wide range of budgets. The gains are more pronounced at moderate to larger budgets, where selection quality plays a greater role, while performance remains stable in the low-budget regime. These results indicate that UCS provides robust performance benefits that could generalize across different demonstration selection budget.

## 6 Ablation Studies

To isolate the impact of core design choices in UCS, we conduct controlled ablations on embedding construction, cluster definition, and scoring design using the Llama model (Table 8).

Table 7: **Sensitivity to selection budget.** Accuracy (%) as a function of the number of demonstrations for MDL and UCS+MDL with Llama on Banking77 dataset.

| Budget | MDL | UCS+MDL | Δ |
|---|---|---|---|
| 0 | 34.7±0.7 | 34.9±0.2 | +0.2 |
| 1 | 56.4±1.1 | 56.5±1.6 | +0.1 |
| 5 | 70.9±1.9 | 72.4±1.1 | +1.5 |
| 10 | 71.8±0.3 | 73.2±1.2 | +1.4 |
| 20 | 63.6±0.3 | 65.9±2.1 | +2.3 |

**Embedding Construction.** Results indicate that UCS is robust to embedding extraction choices, though last-layer representations with mean pooling consistently yield the best results. Mid-layer features and last-token pooling degrade performance, suggesting that aggregated, final-layer semantics best capture latent task structure. Interestingly, applying $\ell_2$ normalization (*Last+Mean+*$\ell_2$*Norm*) offers slight gains over the unnormalized default, indicating that directional consistency aids density-based clustering.

**Cluster Definition.** We compare our dictionary-based clustering against direct embedding clustering without dictionary learning (*dbscan*) and simple atom assignment (*dict_argmax*). UCS outperforms both baselines, particularly on HWU64 ($+2.9\%$ vs. *dbscan*). This confirms that projecting into a sparse dictionary space effectively denoises the representation, while density-based clustering preserves local semantic structures that simple argmax assignment destroys.

**Importance of SGT Scoring.** Replacing the full UCS objective with coverage-only ($n_{\text{seen}}$) leads to significant performance drops (e.g., $-2.8\%$ on HWU64). The superior performance of the combined default objective confirms that the SGT-derived unseen estimator is non-redundant: it guides selection toward both seen and unseen coverage that simple coverage metrics fail to anticipate.

**D. Rarity-Only VoteK Controls.** To test whether UCS+VoteK gains are simply due to rarity weighting, we evaluate two rarity-only controls that keep the VoteK protocol but *remove* the Good–Turing/SGT extrapolation: **B1** (inverse cluster-size rarity) adds a bonus depending only on cluster size, and **B2** (spectrum rarity without GT) uses the size spectrum but explicitly drops the GT ratio/discounting. Table 9 shows that neither control matches UCS+VoteK: the rarity-only baselines are inconsistent across datasets and fail to reproduce UCS+VoteK improvements, supporting that gains come from the SGT-based extrapolation rather than naive rarity weighting.

Table 8: **Ablation results**: Accuracy in %, mean±std over 3 runs on three datasets. "Mid" and "Last" refer to the layer used for embedding extraction. Two alternative clustering methods are described in Section 3.4.

| UCS+DPP | Banking77 | CLINC150 | HWU64 |
|---|---|---|---|
| *Embedding construction* | | | |
| Mid+Mean | 73.9±0.1 | 70.7±0.8 | 57.3±1.2 |
| Last+LastToken | 73.7±1.4 | 71.7±0.5 | 56.7±1.5 |
| Last+Mean+$\ell_2$Norm | 74.8±1.3 | 71.7±0.9 | 57.5±0.8 |
| *Clustering method* | | | |
| dbscan | 73.6±1.1 | 70.4±0.2 | 54.2±0.6 |
| dict_argmax | 74.0±0.7 | 69.6±0.1 | 55.8±1.1 |
| *UCS score design* | | | |
| $n_{\text{seen}}$ | 74.6±0.1 | 70.2±0.7 | 54.3±0.3 |
| $\hat{n}_{\text{unseen}}$ | 74.4±0.6 | 70.3±0.7 | 54.1±0.8 |
| $\hat{n}_{\text{unseen}}/n_{\text{seen}}$ | 74.7±0.5 | 70.1±0.6 | 54.0±0.8 |
| UCS | 74.6±0.6 | 71.4±0.9 | 57.1±0.8 |

Table 9: **Rarity-only VoteK controls** (Llama-3.2-3B-it). Mean±std accuracy (%) over 3 runs. B1: inverse-size rarity. B2: spectrum rarity without GT.

| Method | Banking77 | CLINC150 | HWU64 |
|---|---|---|---|
| VoteK | 42.1±0.9 | 59.7±0.3 | 45.7±0.4 |
| UCS+VoteK | 42.6±0.2 | 59.9±0.8 | 49.8±2.0 |
| B1 | 40.8±2.3 | 60.0±1.4 | 44.2±1.1 |
| B2 | 43.6±1.7 | 59.1±1.1 | 44.2±1.1 |

## 7 Conclusion

We proposed UCS, a statistically grounded framework for improving in-context demonstration selection via a subset-level *coverage* prior. UCS discretizes LLM embeddings into latent clusters and uses a Smoothed Good–Turing estimator to quantify how much seen and unrevealed clusters a candidate subset covers under the same candidate-pool sampling process. Across three intent classification benchmarks, three BBEH reasoning tasks, and three frontier LLMs, UCS consistently improves or matches strong baselines, with the largest gains for query-independent selection and stable improvements for query-dependent selectors. Beyond accuracy gains, UCS offers interpretable insights into cluster distributions across tasks and models, highlighting the long-tailed and model-specific nature of latent cluster distribution. Overall, UCS provides a lightweight, training-free plug-in for more reliable ICL selection by explicitly encouraging broader latent cluster coverage.

## Limitations

First, UCS discretizes continuous LLM embeddings into latent clusters using unsupervised methods. While this design is flexible and training-free, the resulting units may not always align with human-interpretable concepts. Future work could explore alternative or hybrid unit discovery strategies, including supervised signals, semantic grounding, or multi-granular representations.

Second, while we have extended evaluation to three BBEH reasoning tasks, the majority of our experiments focus on intent classification with relatively short demonstrations and contexts. Extending UCS to tasks with longer inputs, generation-heavy objectives, or structured outputs (e.g., mathematical reasoning) remains a promising direction.

Third, UCS is designed to be model-consistent and therefore requires embedding the demonstration pool using the LLM at evaluation time. This incurs additional offline computation for large pools, but also opens up opportunities for future work on shared or aligned representations across multiple models.

Finally, in this work we only combine UCS with widely used and well-established baseline selectors (e.g., VoteK, DPP, MDL) to provide a clear proof of concept. Recent reinforcement-learning-based selection methods offer additional opportunities for integration. Exploring how UCS can be incorporated into policy-based iterative selection pipelines—such as by serving as a coverage-aware reward or regularizer—is an exciting avenue for future research.

## References


Sweta Agrawal, Chunting Zhou, Mike Lewis, Luke Zettlemoyer, and Marjan Ghazvininejad. 2023. In-context examples selection for machine translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8857–8873.

Shengnan An, Bo Zhou, Zeqi Lin, Qiang Fu, Bei Chen, Nanning Zheng, Weizhu Chen, and Jian-Guang Lou. 2023. Skill-based few-shot selection for in-context learning. *arXiv preprint arXiv:2305.14210*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. 2020. Efficient Intent Detection with Dual Sentence Encoders. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 38–45, Online. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, and 1 others. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

Irving J Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264.

Shivanshu Gupta, Matt Gardner, and Sameer Singh. 2023. Coverage-based example selection for in-context learning. *arXiv preprint arXiv:2305.14907*.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Yuanzhu Peter Chen, and 1 others. 2025. Big-bench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26473–26501.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Alex Kulesza, Ben Taskar, and 1 others. 2012. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286.

Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. 2019. An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1311–1316, Hong Kong, China. Association for Computational Linguistics.

Itay Levy, Ben Bogin, and Jonathan Berant. 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)*, pages 1401–1422.

Xiang Li, Jiayi Xin, Qi Long, and Weijie J Su. 2025. Evaluating the unseen capabilities: How

many theorems do llms know? *arXiv preprint arXiv:2506.02058*.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520.

Haoyu Liu, Jianfeng Liu, Shaohan Huang, Yuefeng Zhan, Hao Sun, Weiwei Deng, Furu Wei, and Qi Zhang. 2024. Se^2: Sequential example selection for in-context learning. *arXiv preprint arXiv:2402.13874*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B Dolan, Lawrence Carin, and Weizhu Chen. 2022. 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*, pages 100–114.

Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. 2019. Benchmarking natural language understanding services for building conversational agents. In *Increasing Naturalness and Flexibility in Spoken Dialogue Interaction - 10th International Workshop on Spoken Dialogue Systems, IWSDS 2019, Syracuse, Sicily, Italy, 24-26 April 2019*, volume 714 of *Lecture Notes in Electrical Engineering*, pages 165–183. Springer.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 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)*, pages 8086–8098.

Man Luo, Xin Xu, Zhuyun Dai, Panupong Pasupat, Mehran Kazemi, Chitta Baral, Vaiva Imbrasaite, and Vincent Y Zhao. 2023. Dr. icl: Demonstration-retrieved in-context learning. *arXiv preprint arXiv:2305.14128*.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022a. Metaicl: Learning to learn in context. In *Proceedings of the 2022 conference of the North American chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022b. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*.

Alon Orlitsky, Ananda Theertha Suresh, and Yihong Wu. 2016. Optimal prediction of the number of unseen species. *Proceedings of the National Academy of Sciences*, 113(47):13283–13288.

Keqin Peng, Liang Ding, Yancheng Yuan, Xuebo Liu, Min Zhang, Yuanxin Ouyang, and Dacheng Tao. 2024. Revisiting demonstration selection strategies in in-context learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9090–9101.

Chengwei Qin, Aston Zhang, Chen Chen, Anirudh Dagar, and Wenming Ye. 2024. In-context learning with iterative demonstration selection. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7441–7455, Miami, Florida, USA. Association for Computational Linguistics.

Alexander Scarlatos and Andrew Lan. 2023. RetICL: Sequential retrieval of in-context examples with reinforcement learning. *Preprint*, arXiv:2305.14502.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and 1 others. 2022. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv:2209.01975*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Minh-Hao Van, Xintao Wu, and 1 others. 2024. In-context learning demonstration selection via influence analysis. *arXiv preprint arXiv:2402.11750*.

Liang Wang, Nan Yang, and Furu Wei. 2024. Learning to retrieve in-context examples for large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1752–1767.

Xubin Wang, Jianfei Wu, Yichen Yuan, Deyu Cai, Mingzhe Li, and Weijia Jia. 2025. Demonstration selection for in-context learning via reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*. ICML.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, and 1 others. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhenyu Wu, Yaoxiang Wang, Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Jingjing Xu, and Yu Qiao. 2023a. 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)*, pages 489–498.

Zhiyong Wu, Yaoxiang Wang, Jiacheng Ye, and Lingpeng Kong. 2023b. 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)*, pages 1423–1436.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Zhao Yang, Yuanzhe Zhang, Dianbo Sui, Cao Liu, Jun Zhao, and Kang Liu. 2023. Representative demonstration selection for in-context learning with two-stage determinantal point process. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5443–5456, Singapore. Association for Computational Linguistics.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2023a. Compositional exemplars for in-context learning. In *International Conference on Machine Learning*, pages 39818–39833. PMLR.

Xi Ye, Srinivasan Iyer, Asli Celikyilmaz, Veselin Stoyanov, Greg Durrett, and Ramakanth Pasunuru. 2023b. Complementary explanations for effective in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4469–4484.

Changshuo Zhang, Ang Gao, Xiao Zhang, Yong Liu, and Deyang Li2 Fangchao Liu2 Xinyu Zhang. 2025. Reward mixology: Crafting hybrid signals for reinforcement learning driven in-context learning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 4373–4383.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. In *The eleventh international conference on learning representations*.

# A Additional Details for the UCS Algorithm

## A.1 Embedding Details

**Pooling choice.** In the main paper we use masked mean pooling (Eq. 1) to obtain a single vector per input. We also experimented with alternative pooling rules such as taking the first token embedding or the last non-padding token embedding; these variants produced similar qualitative trends but were slightly less stable across models.

**Layer selection.** Unless otherwise specified, we extract hidden states from a fixed layer $\ell$ of the same LLM used for ICL evaluation. In practice, using either the final layer or a mid-layer works well; we fix $\ell$ across all methods for a fair comparison.

**Normalization.** We do not apply $\ell_2$ normalization in our main experiments. (When enabled, normalization can reduce scale effects for cosine-distance-based clustering, but it was unnecessary in our setup.)

**Optional PCA.** For speed and memory, we optionally apply PCA to reduce embedding dimensionality before downstream clustering/dictionary learning. When enabled, PCA is fit on the training pool embeddings only and applied to the same pool; this does not use labels.

**Batching and throughput notes.** Embedding is computed in mini-batches with truncation to a maximum sequence length. When the embedding model is sharded across multiple devices (e.g., `device_map=auto`), we place input tensors on the device that hosts the token embedding layer to avoid unnecessary transfers. We also disable KV-cache during embedding extraction (since generation is not needed), which improves stability for some architectures.

## A.2 DBSCAN Radius Selection via $k$NN Quantiles

DBSCAN requires a neighborhood radius `eps` and a minimum number of neighbors `min_samples`. We set `eps` *data-adaptively* using a simple $k$NN quantile heuristic that is robust across datasets and embedding scales.

**Cosine distance in normalized code space.** We run DBSCAN on the normalized dictionary codes $\{\bar{r}_i\}_{i=1}^{N}$ (Eq. 4) using cosine distance

$$d_{\cos}(\bar{r}_i, \bar{r}_j) \;=\; 1 - \frac{\langle \bar{r}_i, \bar{r}_j \rangle}{\|\bar{r}_i\|_2 \|\bar{r}_j\|_2}.$$

Since $\bar{r}_i$ is already $\ell_2$-normalized, cosine distance reduces to $1 - \langle \bar{r}_i, \bar{r}_j \rangle$.

**Heuristic for choosing `eps`.** For each point $i$, we compute its $k$-th nearest-neighbor distance under cosine distance:

$$d_i^{(k)} \;=\; k\text{-NN distance of } \bar{r}_i \text{ among } \{\bar{r}_j\}_{j \neq i}.$$

We then set

$$\texttt{eps} \;=\; \mathrm{Quantile}_q\Big(\{d_i^{(k)}\}_{i=1}^{N}\Big), \qquad (13)$$

where $q \in (0, 1)$ is a user-controlled quantile (e.g., $q \in [0.05, 0.15]$ in our sweeps) and $k$ is a small constant (e.g., $k = 10$). Intuitively, $d_i^{(k)}$ estimates a local neighborhood radius around point $i$; taking a low quantile yields an `eps` that is tight enough to avoid merging unrelated patterns, while still admitting dense "head" regions of the code manifold. This heuristic automatically adapts to the intrinsic spread of the codes, removing the need for dataset-specific distance calibration.

**Practical knobs.** In code space, we use a slightly larger `min_samples` than in raw embedding DBSCAN to discourage spurious micro-clusters formed by isolated points (e.g., `min_samples` $\geq 5$). When DBSCAN returns noise points ($c_i = -1$), we map each of them to a new singleton, ensuring every example contributes a valid discrete type count for downstream SGT estimation.

### A.3 Why Dictionary+DBSCAN Rather than Argmax Atom Assignment

A natural discretization after dictionary learning is to assign each example to its largest-magnitude atom,

$$c_i \;=\; \arg\max_{j \in [K]} |r_{ij}|.$$

This *argmax-atoms* scheme is fast and yields exactly $K$ types, but we found it to be a weaker inductive bias for inducing clusters than clustering in code space.

**(1) Argmax collapses compositional structure.** Dictionary codes are not meant to be purely one-hot: many embeddings are better described by *combinations* of atoms. Two examples may share the same dominant atom but differ substantially in the remaining activated atoms; argmax forces them into the same unit even if their latent patterns differ. DBSCAN on the full (normalized) code vectors can separate such cases by grouping on *overall code similarity* rather than a single coordinate.

**(2) Heavy-tailed atom usage distorts type counts.** In practice, atom activations exhibit a heavy-tailed frequency spectrum: a few atoms become "head" atoms used by many points, while many atoms are rarely used. Argmax assignment tends to over-concentrate mass on head atoms, producing a small number of very large clusters and a long tail of tiny clusters. This concentration reduces the effective number of types observed in a subset, which can make unseen estimation less informative under a fixed budget. Clustering in code space mitigates this by allowing multiple distinct clusters to exist within the region dominated by a head atom, as long as the full code patterns differ.

**(3) Better alignment with the "type" abstraction needed by SGT.** SGT-style estimators treat each unit as an abstract type that can be observed multiple times in a sample. A good type system should be (i) stable to small perturbations, (ii) fine-grained enough to distinguish qualitatively different patterns, and (iii) coarse enough that repeated observations occur at realistic subset sizes. Argmax is often too coarse in high-density regions (merging distinct patterns) and too brittle around ties (small coefficient changes can flip the argmax atom). DBSCAN on normalized codes provides a more stable middle ground: it forms repeatable head clusters where patterns recur, and allocates ambiguous or idiosyncratic points to tail clusters or the noise unit.

**Connection to our ridge codes.** We use ridge (L2) coding (Eq. 3) specifically to avoid overly sparse, near-one-hot codes that make argmax assignment almost deterministic and distance computations brittle. With smoother ridge codes, cosine distances in code space become meaningful, which improves DBSCAN's ability to discover consistent latent units.

**Takeaway.** Argmax-atoms is a useful baseline discretization, but dictionary+DBSCAN better respects compositional cluster signatures and yields a unit system that empirically supports stronger coverage estimates and downstream selection.

### A.4 Smoothed Good–Turing Estimator Details

This appendix specifies the stabilized Smoothed Good–Turing (SGT) procedure used to compute $\widehat{U}_t(S)$ from the subset spectrum $\{f_s(S)\}_{s \geq 1}$. We follow the classical Good–Turing estimator but incorporate standard finite-sample stabilizations: (i)

truncation/binning of high counts, (ii) an offset to avoid unstable extrapolation, and (iii) optional smoothing of the frequency spectrum.

**Inputs.** Given a candidate subset $S$, we compute the multiplicities $n_u(S)$ and the spectrum $f_s(S) = |\{u : n_u(S) = s\}|$ for $s \geq 1$. We optionally exclude a designated noise label (e.g., DBSCAN noise) from the type universe before forming $\{f_s(S)\}$.

**Classical Good–Turing functional.** Let $m > 0$ denote the number of additional samples we wish to extrapolate, and define the expansion factor $t := m/|S|$ (e.g., $t = 5$ means extrapolating to $5 \times |S|$ additional draws). The (unsmoothed) Good–Turing estimator for the number of unseen types after $m$ additional draws is

$$\widehat{U}_t^{\mathrm{GT}}(S) = -\sum_{s\geq 1}(-t)^s\, f_s(S). \tag{14}$$

In finite samples, the alternating series in (14) can have high variance for moderate/large $t$, motivating stabilized variants.

**Truncation / binning.** To reduce sensitivity to large multiplicities (which are typically sparse), we truncate the spectrum at a maximum count $M$ (denoted `bin_size` in code):

$$\widehat{U}_{t,M}^{\mathrm{GT}}(S) = -\sum_{s=1}^{M}(-t)^s\, f_s(S), \tag{15}$$

treating $f_s(S) = 0$ for $s > M$. We use a small default $M$ (e.g., 20) throughout experiments, and report sensitivity in ablations.

**Offset-stabilized extrapolation.** Following standard stabilization heuristics, we introduce an *offset* parameter $\alpha \in [1, 2]$ (denoted `offset` in code) that damps the contribution of higher-order terms for extrapolation. Concretely, we reweight each term in (15) by a tail probability factor $w_s(t, \alpha) \in [0, 1]$ that decreases with $s$:

$$\widehat{U}_{t,M}^{\mathrm{SGT}}(S) = -\sum_{s=1}^{M}(-t)^s\, w_s(t,\alpha)\, f_s(S). \tag{16}$$

Intuitively, $w_s(t, \alpha)$ acts as a soft truncation that discounts unstable high-order terms when extrapolating beyond the observed sample. In our implementation, $w_s(t, \alpha)$ is computed via a simple binomial tail expression (as in `utils_sgt.py`), and we fix $\alpha$ to a default value (e.g., $\alpha = 1.0$) unless stated otherwise.

**Optional spectrum smoothing.** The raw spectrum $\{f_s(S)\}$ can be noisy, especially for small subsets. Optionally, we smooth the nonzero portion of the spectrum by fitting a simple parametric model (e.g., power-law) and replacing $f_s(S)$ by a smoothed $\widetilde{f}_s(S)$ before applying (16). When enabled, we fit the model to the first $K$ bins (default $K = M$) and clamp negative fitted values to zero:

$$f_s(S) \;\leftarrow\; \widetilde{f}_s(S), \qquad s = 1, \ldots, M. \tag{17}$$

This option is primarily used as a robustness knob; our main experiments use smoothing only when explicitly noted.

**Non-negativity and numerical guards.** Since $\widehat{U}_t(S)$ represents an unseen *count*, we enforce non-negativity:

$$\widehat{U}_t(S) = \max\big(0,\ \widehat{U}_{t,M}^{\mathrm{SGT}}(S)\big). \tag{18}$$

If the estimate is non-finite (NaN/Inf) due to numerical issues, we also set it to zero. These guards match our selection-time implementation, ensuring that the UCS score $\widehat{K}_t(S) = K_{\mathrm{seen}}(S) + \widehat{U}_t(S)$ remains well-defined for all subsets.

**Hyperparameters.** We summarize the estimator hyperparameters used throughout the paper:

- Expansion factor: $t$ (default $t = 5$).
- Truncation / bin size: $M$ (default $M = 20$).
- Offset: $\alpha \in [1, 2]$ (default $\alpha = 1.0$).
- Optional spectrum smoothing: on/off (default off unless stated).

We either fix these values across datasets or tune lightly on a small validation split; details are provided alongside experimental settings.

## B Runtime Breakdown

Table 10 reports the one-time offline preprocessing cost (embedding + dictionary learning + DBSCAN), and Table 11 reports the end-to-end online overhead (selection + ICL evaluation) when adding UCS. All experiments use a single NVIDIA A100 80GB GPU. For our recommended pairings (MDL+UCS and VoteK+UCS), the additional end-to-end time is typically ∼0–3 s. DPP+UCS incurs larger overhead (+600–680 s) due to recomputation inside the greedy loop; we treat MDL/VoteK+UCS as the default efficient instantiations.

Table 10: **Offline preprocessing** (one-time per dataset+model, in seconds).

| Dataset (size) | Pool embed | Dict fit | Dict transform | DBSCAN | Total |
|---|---|---|---|---|---|
| CLINC150 (∼15k) | 35.88 | 19.65 | 0.17 | 0.37 | 57.03 |
| HWU64 (∼10k) | 19.87 | 16.57 | 0.24 | 0.24 | 37.83 |
| BANKING77 (∼11k) | 27.70 | 14.25 | 0.19 | 0.26 | 43.31 |

Table 11: **Online selection overhead** ($\Delta$ end-to-end time in seconds, selection + ICL evaluation).

| Dataset | MDL+UCS | VoteK+UCS | DPP+UCS |
|---|---|---|---|
| CLINC150 | +1.96 | −13.37 | +605.84 |
| HWU64 | −0.09 | +2.75 | +680.70 |
| BANKING77 | +2.28 | +1.19 | +625.38 |

# C Details of UCS Instantiations

## C.1 Cluster assignments and frequency-of-frequencies

Each training example $i$ is assigned to a discrete cluster $c(i) \in \mathcal{C}$ as described in Section 3.4. A designated noise label (e.g., DBSCAN noise) is excluded from all UCS computations below.

We use two types of frequency summaries.

**Subset-level frequency-of-frequencies (DPP and MDL).** For a candidate subset $S$, let

$$n_u(S) = \big|\{\, i \in S : c(i) = u \,\}\big| \tag{19}$$

denote the multiplicity of cluster $u$ in $S$. The corresponding frequency-of-frequencies (FoF) is defined as

$$f_s(S) = \big|\{\, u : n_u(S) = s \,\}\big|. \tag{20}$$

**Corpus-level cluster-size spectrum (VoteK).** Over the entire training pool, let

$$n_u = \big|\{\, i : c(i) = u \,\}\big| \tag{21}$$

denote the global size of cluster $u$. We define the corpus-level frequency-of-frequencies (FoF) as

$$g_s = \big|\{\, u : n_u = s \,\}\big|. \tag{22}$$

The spectrum $\{g_s\}$ characterizes the global distribution of cluster sizes in the training corpus.

## C.2 Subset-level UCS for DPP and MDL

For query-dependent selectors (DPP and MDL), UCS operates at the subset level.

**UCS coverage functional.** Given a subset $S$, we define the UCS coverage score as

$$\Phi(S) = K_{\text{seen}}(S) + \widehat{U}_t(S), \tag{23}$$

where

$$K_{\text{seen}}(S) = \big|\{\, u : n_u(S) > 0 \,\}\big| \tag{24}$$

is the number of distinct clusters observed in $S$. The term $\widehat{U}_t(S)$ estimates the number of previously unseen clusters that would be revealed after observing $t > 0$ additional samples drawn from the same underlying process as $S$. This estimate is computed from the subset-level frequency-of-frequencies $\{f_s(S)\}_{s\geq 1}$ using a stabilized Smoothed Good–Turing estimator. Negative or non-finite estimates are clamped to zero for numerical stability.

Intuitively, $\Phi(S)$ favors subsets that both cover many distinct clusters and exhibit frequency patterns indicative of additional latent coverage.

**DPP + UCS.** For DPP, demonstrations are selected greedily. At each step, we select the example $i \notin S$ that maximizes

$$\Delta_{\text{DPP}}(i \mid S) + \lambda\big(\Phi(S \cup \{i\}) - \Phi(S)\big), \tag{25}$$

where $\Delta_{\text{DPP}}(i \mid S)$ is the standard DPP marginal gain. The UCS term therefore rewards candidates that substantially increase the estimated coverage of the growing subset.

**MDL + UCS.** For MDL, we score complete candidate subsets rather than individual additions. Each subset $S$ is assigned the score

$$\text{MDL}(S; x) + \lambda\, \Phi(S), \tag{26}$$

where $\text{MDL}(S; x)$ is the original MDL objective based on label uncertainty. UCS thus acts as a subset-level regularizer, encouraging MDL to prefer subsets with broader estimated coverage. Candidate subsets are generated using the same proposal mechanism as the original MDL retriever.

## C.3 Corpus-level UCS prior for VoteK

VoteK selects a single, query-independent demonstration set. Accordingly, UCS is instantiated as a factorized corpus-level rarity prior derived via Smoothed Good-Turing (SGT) estimation.

**SGT Prior Computation.** We first compute the corpus-level FoF spectrum $\{g_s\}_{s\geq 1}$. To estimate the probability mass of clusters with size $s$, we apply the SGT estimator:

1. **Smoothing:** We fit a parametric model (e.g., Power-law, Poisson, or Negative Binomial) to $\{g_s\}$ to obtain a non-negative smoothed spectrum $\{\widehat{g}_s\}_{s\geq 1}$.

2. **Good-Turing Adjustment:** We compute the adjusted count $s^*$ for size $s$ using the Good-Turing rule:

$$s^* = (s+1)\frac{\widehat{g}_{s+1}}{\widehat{g}_s}. \tag{27}$$

3. **Weight Assignment:** The estimated probability mass for a cluster of size $s$ is given by $p(s) = s^*/N$, where $N$ is the total number of examples. Each cluster $u$ is assigned a weight inversely proportional to this mass:

$$w_u \propto \frac{1}{p(n_u) + \varepsilon}, \tag{28}$$

where $n_u$ is the size of cluster $u$ and $\varepsilon$ ensures stability. Weights are normalized such that their mean over non-ignored clusters is one.

**VoteK + UCS scoring.** Given the standard VoteK vote count $v(i)$ for example $i$, we define the final score as

$$\text{score}(i) = v(i) + \lambda \log w_{c(i)}. \tag{29}$$

When $\lambda = 0$, this recovers the original VoteK ranking. For $\lambda > 0$, the additional term promotes examples from clusters with globally rare sizes (as estimated by SGT), enhancing the diversity of the selected corpus-level demonstration set.

## D Hyperparameter Settings for Reproducibility

For reproducibility, we report the complete set of hyperparameters used in our experiments. Table 12 summarizes the common settings shared across all experiments, including budget, batch sizes, and dictionary learning configurations.

While most parameters are fixed, we tune two key hyperparameters using grid search to adapt to the semantic density of different embedding spaces and datasets:

- `dbscan_q`: The density quantile threshold for the dictionary-based DBSCAN clustering. This determines the granularity of the discovered clusters. Tuned values are reported in Table 13.
- `sgt_lambda` ($\lambda$): The weight of the UCS unseen-coverage term added to the Retriever's score. This balances the trade-off between the base objective (e.g., diversity in DPP, compression in MDL) and the exploration of rare/unseen clusters. Tuned values for each retriever-model-dataset combination are listed in Table 14.

Unless explicitly varied in ablation studies, all other settings follow Table 12.

Table 12: **Common Hyperparameters used for Experiments.**

| Component | Hyperparameter | Value |
|---|---|---|
| Data / Evaluation | -dataset_type | banking77 / clinc150 / hwu64 |
| | -budget | 10 |
| | -test_size | 500 |
| | -n_runs | 3 |
| | -seed | 42 |
| | -icl_batch_size | 16 |
| Backbone / Embeddings | -endpoint_name | Qwen/Qwen2.5-7B-Instruct, meta-llama/Llama-3.2-3B-Instruct, google/gemma-2-9b-it |
| | -embedding_model_name | Qwen/Qwen2.5-7B-Instruct, meta-llama/Llama-3.2-3B-Instruct, google/gemma-2-9b-it |
| | -sentence_transformers_model | all-mpnet-base-v2 |
| | -candidate_num | 50 |
| Dictionary Learning | -dict_n_components | 64 |
| | -dict_alpha | 10.0 |
| | -dict_top_k | 4 |
| | -dict_tau | $10^{-3}$ |
| | -dict_transform_algorithm | omp |
| | -dict_regularization_type | l2 |
| | -dict_max_iter | 50 |
| | -dict_pca_dim | 128 |
| | -dict_batch_size | 512 |
| Clustering (dict_dbscan) | -clustering | dict_dbscan |
| | -dbscan_k | 20 |
| | -dbscan_q | 0.01 |
| | -dbscan_min_samples | 1 |
| SGT / UCS | -sgt_lambda | 0.1 |
| | -sgt_t | 5 |
| | -sgt_bin_size | 20 |
| | -sgt_offset | 1.0 |
| Base Selectors | -baselines | mdl / mdl_sgt / dpp / dpp_sgt / votek / votek_sgt |
| | -votek_k | 3 |
| | -mdl_select_time | 5 |
| | -mdl_ce_model | gpt2 |
| | -dpp_scale_factor | 0.1 |

Table 13: **Tuned `dbscan_q` values.** This parameter controls the density quantile for clustering. It is tuned separately for each backbone and retriever combination to ensure the granularity of clusters matches the retriever's selection logic.

| Backbone | Retriever | Banking77 | CLINC150 | HWU64 |
|---|---|---|---|---|
| Qwen2.5 7B-it | UCS+MDL | 0.15 | 0.01 | 0.15 |
| | UCS+DPP | 0.01 | 0.01 | 0.01 |
| | UCS+VoteK | 0.01 | 0.01 | 0.01 |
| Llama-3.2 3B-it | UCS+MDL | 0.01 | 0.01 | 0.01 |
| | UCS+DPP | 0.01 | 0.01 | 0.01 |
| | UCS+VoteK | 0.01 | 0.01 | 0.01 |
| Gemma-2 9B-it | UCS+MDL | 0.01 | 0.01 | 0.15 |
| | UCS+DPP | 0.01 | 0.10 | 0.01 |
| | UCS+VoteK | 0.01 | 0.01 | 0.5 |

Table 14: **Tuned `sgt_lambda` values.** This parameter controls the strength of the UCS term ($\lambda$) added to the Retriever score. It balances the exploitation of the base method (MDL/DPP/VoteK) with the exploration of unseen clusters.

| Backbone | Retriever | Banking77 | CLINC150 | HWU64 |
|---|---|---|---|---|
| Qwen2.5 7B-it | UCS+MDL | 0.01 | 0.005 | 0.001 |
| | UCS+DPP | 0.001 | 0.01 | 0.01 |
| | UCS+VoteK | 5.0 | 5.0 | 5.0 |
| Llama-3.2 3B-it | UCS+MDL | 0.05 | 0.05 | 0.05 |
| | UCS+DPP | 0.50 | 0.05 | 0.05 |
| | UCS+VoteK | 0.05 | 0.05 | 0.50 |
| Gemma-2 9B-it | UCS+MDL | 0.05 | 0.05 | 0.05 |
| | UCS+DPP | 0.50 | 0.10 | 0.05 |
| | UCS+VoteK | 0.50 | 0.10 | 0.05 |