Agentic Clustering: Controllable Text Taxonomies via Multi-Agent Refinement

Anonymous ACL submission

Abstract

Recent text-clustering methods use large language models to propose a cluster taxonomy from a corpus and then assign each text to it. These pipelines are fundamentally programmatic: the sequence of LLM calls and the rules for stopping, merging, and splitting clusters are fixed in code in advance, so they generalise poorly across corpora of different structure and cannot easily incorporate user-supplied constraints such as a target cluster count or a clustering intent. We propose an agentic alternative in which an orchestrator LLM inspects the state of the discovery process at each step and dispatches one of a small set of specialised agents — proposer, synthesizer, auditor, investigator, and critic — adapting the pipeline to the corpus rather than executing a fixed one. On seven public text-clustering benchmarks the method achieves state-of-the-art performance, beating the strongest prior LLM baseline by up to 32% in ARI.

1 Introduction

A common task in exploring large unstructured text corpora is clustering the texts into meaningful groups. Clustering approaches could be applied for generating themes from customer feedback, discovering emergent trends in job advertisements, or identifying common themes in scientific literature. Further, a text corpus does not always have a single, obvious clustering, and may have many such groups. For example, a corpus of academic abstracts could be grouped by topic, methodology, or dataset. We refer to these as clustering 'intents'.

Large language models (LLMs) have led to improvements across many NLP tasks. However, the easy application of LLMs to text clustering is limited by the fact that a whole dataset cannot generally fit into the context window of any LLM.

Recent work has focused on using LLMs for clustering by splitting the task into two separate sub-tasks. First, an LLM is fed samples of the dataset, usually in chunks, and generates a taxonomy of cluster names. Second, texts are assigned to a label from this taxonomy, using the taxonomy as the LLM prompt — a standard classification task (Pham et al., 2024; Wang et al., 2023; Huang and He, 2024). The most recent of these methods add further passes that revise, merge or split clusters once an initial draft has been formed (Wan et al., 2024). However, every method in this line is fundamentally *programmatic*: the sequence of LLM calls, the rules for when to stop, merge or split, and the kinds of evidence each step is allowed to consult are all fixed in code in advance. As a consequence, these pipelines are tuned to the datasets they were designed on and do not generalise well to corpora with different structure — in particular, they are inflexible in the granularity of the clusters they produce, performing worse than dense embedding approaches when faced with a dataset of a different granularity.

We propose instead an *agentic* approach to text clustering. Rather than executing a fixed pipeline, an orchestrator agent inspects the state of the discovery process at each step and dispatches one of a small set of specialised agents: a *proposer* that drafts candidate clusters from samples of the corpus, a *synthesizer* that merges those drafts into a single taxonomy, an *auditor* that probes the taxonomy against previously unseen texts, an *investigator* that resolves specific concerns by issuing targeted retrieval queries, and a *critic* that evaluates the taxonomy along several axes. The pipeline is no longer hardcoded but assembled on the fly.

This agentic framing matters because clustering is a problem with no single right answer. The right number of clusters, the right level of abstraction, and the right way to resolve a boundary case all depend on the corpus, and a programmatic pipeline must commit to a fixed answer for each of these in advance. An agentic pipeline can instead interro-

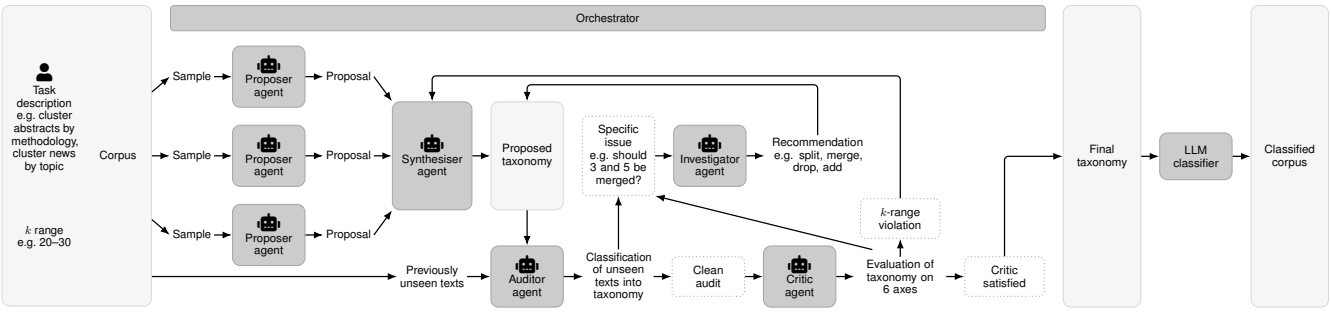

Figure 1: Architecture of the agentic-clustering pipeline.

gate the corpus, observe what the data actually supports, and steer itself accordingly; the same agent capabilities apply unchanged to a 77-class banking-intent corpus and an 18-class spoken-domain corpus, with the orchestrator adapting its dispatches to the structure each one exhibits. The same flexibility lets the orchestrator accept user constraints — a target k-range and an optional free-text task description of the clustering lens — and forward them to every agent without rewriting the pipeline.

We evaluate our approach on seven public text-clustering benchmarks, and find that it achieves state-of-the-art performance, exceeding prior approaches by up to 32% in ARI score.

We release this approach as a Claude Code skill, for easy deployment, under the MIT License.[1]

2 Related work

Probabilistic topic models. Traditional probabilistic topic models are still widely used for text clustering, particularly in the social sciences and humanities. This line of work treats each document as a mixture over latent topics and each topic as a distribution over words, recovering both from a bag-of-words representation via variational EM or Gibbs sampling. Latent Dirichlet Allocation (Blei et al., 2003) is the canonical example.[2] These approaches are limited by relying on a bag-of-words document representation, and by the need to pre-specify the number of clusters.

Dense embeddings with classical clustering. The advent of transformers (Vaswani et al., 2017) led to a new paradigm for text clustering, where a transformer is used to encode the documents into a dense embedding vector, and then classical clustering algorithms (e.g. k-means, HDBSCAN, etc.) are used to cluster the vectors. Dense embeddings generally outperform probabilistic topic models. Sentence-BERT (Reimers and Gurevych, 2019) fine-tunes BERT so that semantically similar sentences land nearby in vector space, and k-means on top yields semantic clusters at a pre-specified k. BERTopic (Grootendorst, 2022) adds UMAP dimensionality reduction and HDBSCAN, allowing variable k; Top2Vec (Angelov, 2020) jointly learns document and word vectors in the same UMAP+HDBSCAN regime.

LLM-improved dense embeddings. Two lines of work use LLMs to improve dense-embedding clustering. The first shapes the embedding space itself: ClusterLLM (Zhang et al., 2023) fine-tunes the encoder from LLM-answered triplet queries and uses pairwise prompts to set hierarchical granularity; Viswanathan et al. (2024) apply LLMs to enrich inputs, supply pairwise constraints, and post-correct assignments; IDAS (De Raedt et al., 2023) clusters utterances jointly with LLM-written labels; and LLMEdgeRefine (Feng et al., 2024) re-assigns only low-confidence boundary points. The second replaces the encoder with an LLM embedder: Petukhova et al. (2025) show this beats BERT-based baselines.

LLM-driven taxonomy generation. A further line of work has focused on using LLMs to generate the taxonomy itself. PromptTopic (Wang et al., 2023) runs every text through an LLM to generate a candidate topic, and then uses an LLM to merge all the candidate topics into a final taxonomy. Huang and He (2024) follows a similar approach, but passes the corpus to the LLM in batches of 15 texts at a time. They then use a lightweight classifier to assign each text to a topic. Passing all texts through an LLM may be infeasible for a large corpus, so TopicGPT (Pham et al., 2024) passes

[1] The released artifact ships only the agent prompts and orchestration code, with no data derived from the research-licensed benchmarks used for evaluation.

[2] See Jagarlamudi et al. (2012) and Akash et al. (2024) for extensions.

the texts one at a time, but asks it to compare to a growing taxonomy of topics, with early stopping if 200 texts are processed with no addition to the taxonomy. They then also follow this with a merging step, and a lightweight classifier.

Building on these approaches, the most recent line of work has added further refinement stages into taxonomy generation. TnT-LLM (Wan et al., 2024) uses an LLM to summarize a sample of documents, and splits this sample into minibatches. They use the first mini-batch to initialise a taxonomy, and then subsequent minibatches to refine the taxonomy, adding, splitting or merging clusters. Additionally, unlike the previous methods, TnT-LLM allows for user-specified cluster intents.

3 Method

Our method replaces the fixed pipeline of prior LLM-driven taxonomy work with a set of specialised agents coordinated by an orchestrator that dispatches them in a state-dependent order. The agent capabilities themselves recall earlier work — the proposer samples corpus batches as in Huang and He (2024), the synthesizer merges drafts as in Wang et al. (2023), and the auditor, investigator and critic refine the output along the lines of Wan et al. (2024) — but the orchestrator decides which to dispatch, when, and what to do with each return value, removing the hardcoded stop, merge and split rules that constrain those prior pipelines. The framework also accepts user-specified granularity and clustering intent as inputs that the orchestrator forwards to every agent. We find that this combination beats every available baseline on every benchmark we evaluate.

3.1 Inputs

The method takes as input an unlabelled corpus D of texts together with two user constraints. The first is a target *k-range* $[k_{\min}, k_{\max}]$ on the number of clusters, which the synthesizer respects when merging proposals and which the critic checks at review time. The second is an optional free-text *task description* stating the lens through which clusters should be formed (e.g. "cluster by issue type" or "focus on actionable categories for a support team"). When supplied, the task description is forwarded verbatim to every agent and acts as the primary constraint when resolving ambiguous cases; when omitted, agents discover clusters from the data alone. The two constraints operate on orthogonal axes — the task description shapes *what* counts as a cluster, the k-range shapes *how many*.

3.2 Orchestrator

The discovery loop is driven by an *orchestrator* that, at each step, inspects the current workspace state and dispatches one of the agents described in the subsections below. The orchestrator does not itself read the corpus or modify the taxonomy: those actions are reserved for the agents. It also owns the decision of what to do with each agent's return value — whether to apply an investigator's recommendation, how to route a critic's finding, or when to declare the taxonomy final — and is the only component that persists across the entire run.

3.3 Proposer

The proposer is the agent responsible for generating candidate cluster hypotheses directly from the corpus. Each proposer reads a sample of texts and emits a draft cluster set — a list of named clusters with short natural-language descriptions, together with the identifiers of the texts it took as evidence for each cluster and the identifiers of any texts it could not place. Multiple proposers are dispatched in parallel so that the downstream synthesis stage can identify which structure is robust across independent looks at the corpus rather than relying on any single draw. The number of proposers is determined dynamically by the orchestrator: it begins with two or three from different stylistic angles, with additional proposers dispatched when the requested k-range is wide.

Proposers draw from D via a sequential disjoint sampling scheme: each proposer call returns a uniform random subset of texts that have not been returned by any previous call in the same run, and the sampled ids are added to a shared seen set so that subsequent proposers do not see them again. The size of each sample is chosen by the orchestrator at dispatch time and scales inversely with the typical text length — short utterances are sampled in batches of several hundred while longer documents are sampled in batches of tens — so that each proposer sees enough corpus to identify recurring structure while keeping the sample within its working context.

The proposers share a single base prompt and the single user task description but differ in a per-dispatch *style* directive that biases the granularity or framing at which they look for clusters. Example styles include a *balanced* angle that targets the

middle of the k-range, a *fine-grained* angle that pushes for narrower, more specific categories, and a *broad* angle that pushes for fewer, more inclusive ones. The style directive is advisory — it shapes the granularity each proposer aims for but does not override the user task description or the k-range. The motivation is to surface natural structure in the corpus at multiple granularities simultaneously and let the synthesizer arbitrate, rather than committing the system to a single granularity in advance.

Each proposer's output is a structured cluster set: for every proposed cluster, a short name, a one- to two-sentence description, and the ids of the texts that the proposer took as evidence; unplaced ids and any sample-level observations are recorded separately. Proposer outputs are persisted verbatim and consumed in full by the downstream synthesis stage.

3.4 Synthesizer

Once the proposer phase has produced a set of independent draft cluster sets, the synthesizer is dispatched to merge them into a single proposed taxonomy. The synthesizer reads every proposer's output in full, identifies which clusters across proposers refer to the same underlying concept, and emits a proposed taxonomy in which each cluster is named, described, and supported by evidence drawn from the proposer outputs that contributed to it.

The synthesizer relies on two signals to decide which proposed clusters to merge. The first is overlap in the natural-language content of the proposals — clusters with similar names and descriptions across proposers are candidates for merging. The second is direct inspection of the underlying texts: because proposers draw disjoint samples, the same text never appears in two proposals, so the synthesizer instead fetches a handful of evidence texts from each candidate cluster by id and reads them to confirm that the two definitions actually describe the same kind of text. The proposed taxonomy is constrained to lie within the user-specified k-range $[k_{\min}, k_{\max}]$; the synthesizer's merge decisions are biased toward fewer or more clusters as needed to fall inside this range. Clusters that appear in only one proposer's output are retained but flagged as lower-confidence, since they have only been observed in a single draw of the corpus.

Cluster names and descriptions are written under a set of *description quality* rules: descriptions must define the concept that a cluster captures rather than summarise the observed evidence texts; they may not cite specific text identifiers, statistics, or numeric figures from the corpus; and any boundary clarification must be phrased as a generalisable principle rather than an instance-specific correction. The intent is that a reader unfamiliar with the corpus could use the description to assign new texts to the cluster correctly without seeing any of the evidence.

The synthesizer's primary output is a proposed taxonomy: for each cluster, a name, a one- to two-sentence description, the union of evidence text identifiers from the proposer outputs that contributed to it, and a confidence label indicating whether the cluster was supported by multiple proposers or by only one. The synthesizer also emits a separate reasoning trace recording the merge decisions made, the clusters unique to a single proposer that were retained, any conflicts resolved between competing proposals, and the text identifiers that were inspected to verify each merge.

3.5 Auditor

The auditor validates the proposed taxonomy against texts that no agent has yet seen. Where the proposer reasoned from a random draw of the corpus and the synthesizer reasoned over the proposer outputs alone, the auditor confronts the proposed taxonomy with fresh evidence: it pulls a new sample of texts and tests, text by text, whether each can be assigned to one of the proposed clusters.

The auditor draws from the same sequential disjoint sampling regime as the proposer. Each call returns a uniform random subset of texts that have not been returned by any previous call in the same run, including any previous proposer or auditor call. The auditor therefore sees only texts that contributed neither to the proposers' draft cluster sets nor to the synthesizer's merge decisions, so any cluster that holds up under audit holds up on evidence outside its own derivation.

For each text in the sample the auditor produces a single assignment: the id of the best-fitting cluster in the proposed taxonomy together with an integer *confidence score* from 1 (forced guess) to 5 (obvious fit). When no cluster is a defensible fit, the auditor instead assigns the special label *none*; these assignments are recorded without a confidence score and treated separately during aggregation.

The auditor's output combines a per-text assignment table with a set of aggregate statistics: the fraction of texts assigned versus left unclustered

(the *coverage*), mean confidence overall and per cluster, and a list of clusters whose mean assigned confidence falls below a threshold. Clusters on this list are flagged as *weak* and become the natural targets for follow-up by the agents introduced in subsequent subsections.

3.6 Investigator

If the auditor (or later the critic) highlights issues with the taxonomy, the investigator is dispatched. The investigator targets a specific concern (e.g. "should c3 and c5 be merged?") and decides whether the taxonomy should be modified in response.

To gather the texts relevant to its question, the investigator issues one or more query-driven retrieval calls against the corpus: TF-IDF similarity against a free-text query when the concern is conceptual, the texts that recent audits assigned to a particular cluster when the concern is internal coherence, or specific text identifiers when verifying a candidate merge or split. It reads the retrieved texts directly, weighs them against the proposed taxonomy and the user task description, and forms a structured recommendation.

Recommendations take one of six forms: *merge* two or more clusters into one, *split* a cluster into several, *rename* a cluster (and rewrite its description), *add* a new cluster, *remove* a cluster, or *no_change* when no modification is warranted. Each recommendation carries the supporting evidence used to derive it — the search queries issued, the text identifiers inspected, and the investigator's reasoning — together with a confidence level. A *no_change* recommendation additionally records the hypothesis that was rejected, so that subsequent agents do not re-investigate the same question.

The investigator's recommendation is passed back to the orchestrator, which decides whether to implement it. Structural changes (*merge*, *split*, *add*, *remove*) invalidate the existing audit metrics, so an audit pass is naturally re-triggered to validate the modified taxonomy on fresh evidence.

3.7 Critic

The critic is dispatched once the auditor returns a clean audit. Whereas the auditor asks whether the existing clusters can account for previously unseen texts, the critic evaluates the taxonomy against a fixed six-axis checklist, scoring each axis as a pass, warning, or failure:

- *Overlap*: do any two clusters describe the same kind of text?
- *Gaps*: are there obvious categories not covered by any cluster?
- *Granularity*: are some clusters substantially broader or narrower than others?
- *Boundaries*: can similar-sounding clusters be distinguished from their descriptions alone, without consulting the data?
- *Descriptions*: do the cluster descriptions define generalisable concepts rather than overfit to specific evidence?
- *k-range*: does the cluster count fall inside $[k_{\min}, k_{\max}]$?

For axes that require empirical evidence, the critic pulls targeted texts from the corpus: similarity-based retrieval to probe boundary cases between similar-looking clusters, and audit-based retrieval to inspect unclustered patterns when looking for gaps. The user task description, where supplied, defines what counts as a good cluster set: a critique that the clusters are technically coherent but ill-suited to the user's stated purpose is itself a critical finding. The critic's output is a ranked list of issues.

The critic's findings drive the orchestrator's subsequent dispatches. *Overlap*, *granularity*, *boundary* and *gap* issues typically become questions for the investigator; *description*-quality issues prompt the orchestrator to rewrite the affected descriptions directly, without dispatching another agent; and *k-range* violations trigger a fresh synthesis pass. When all six axes pass, the proposed taxonomy is declared final.

3.8 Classification

Once the taxonomy is final, the cluster definitions are applied to the full corpus. Each text in D is sent independently to a language model together with a structured-output schema that constrains the response to one of the taxonomy's cluster identifiers or the special value *none*; the model returns this identifier together with a confidence score and a short reasoning trace.

The classification system prompt is built once per run from the finalised taxonomy. It contains each cluster's name and description but not the example texts that were collected as evidence during discovery, so the classifier must generalise from the

Dataset	Data type	Cluster type	N	k	Not in cluster	Mean chars
Banking77	banking queries	intent	3,080	77	0.0%	54
CLINC150	assistant queries	intent	5,500	150	18.2%	40
MASSIVE-Intent	assistant queries	intent	2,974	60	0.0%	35
MASSIVE-Domain	assistant queries	domain	2,974	18	0.0%	35
GoEmotions	Reddit comments	emotion	45,446	27	35.3%	67
20 Newsgroups	Usenet posts	topic	18,331	20	0.0%	1200
StackExchange	forum titles	topic (forum)	4,156	121	0.0%	57

Table 1: Benchmark datasets.

definitions alone rather than retrieving observed instances. The system prompt is held constant across every text and marked for prompt caching, so the per-text inference cost is dominated by the short user message containing the text under classification; classification can be run synchronously when latency matters or via a batch interface when cost matters more.

An optional refinement step allows the classification prompt to be tuned against a small human-labelled subset of the corpus. A handful of variant prompt headers are scored by classifying the labelled subset under each header and computing agreement with the human labels; the best-scoring header is retained as the classification prompt for the full-corpus pass.

4 Datasets

We evaluate on seven public text-clustering benchmarks (Table 1): Banking77 (Casanueva et al., 2020), CLINC150 (Larson et al., 2019), MASSIVE-Intent and MASSIVE-Domain (FitzGerald et al., 2023), GoEmotions (Demszky et al., 2020), 20 Newsgroups (Lang, 1995), and StackExchange (Zhang et al., 2023). Five are short-text utterance corpora — banking and assistant queries and Reddit comments — that recent LLM-based methods have re-purposed for clustering (Zhang et al., 2023); the remaining two, 20 Newsgroups and StackExchange, exercise the long-document and many-cluster regimes. Two of the seven include a gold "none" class — CLINC's out-of-scope queries and GoEmotions' neutral category — which we deliberately retain so that the benchmark can credit methods that natively express "no cluster." Appendix A documents the exact source, config, split, and cleaning recipe for each dataset.

5 Experimental setup

Baseline selection. We compare against seven prior methods. We choose LDA (Blei et al., 2003) as a representative member of probabilistic topic models. For dense embedding models, we choose Sentence-BERT (Reimers and Gurevych, 2019) + k-means, and BERTopic (Grootendorst, 2022). We also use LLM embeddings + k-means (Petukhova et al., 2025), and ClusterLLM (Zhang et al., 2023), as LLM-based dense embedding refinements. For LLM-driven taxonomy generation methods, we compare with TopicGPT (Pham et al., 2024) and Huang and He (2024). We do not use TnT-LLM (Wan et al., 2024), as the authors did not release their code.

For all cases using LLMs, for a fair comparison, we use Claude Opus 4.7 for 'heavy-weight' tasks that run on small samples of the dataset, such as taxonomy generation, merging and refinement. We use GPT-5-mini for 'light-weight' tasks over the whole corpus, such as classification into the taxonomy. Therefore we reimplement all methods directly. Per-method implementation settings are presented in Appendix B.

k handling. Some methods require the number of clusters, k, to be passed as an input, while others discover k. For methods that require k, we pass the true k. We report results separately for given-k and discover-k methods. Our method and BERTopic can handle both, and so we report results for both. For discover-k for our method, we set a 20% window around the gold count, rounded to integers, so the synthesizer has meaningful freedom to over- or under-estimate while still anchored to the dataset's true granularity.

Noise / out-of-cluster handling. Some datasets include a noise/out-of-cluster class, which is removed by some papers in their evaluation. We choose to include it in our evaluation, so as to be more representative of the real-world use case. For

Method	Banking77				CLINC150				MASSIVE-Intent				MASSIVE-Domain			
	$\hat{k}$	ARI	NMI	ACC	$\hat{k}$	ARI	NMI	ACC	$\hat{k}$	ARI	NMI	ACC	$\hat{k}$	ARI	NMI	ACC
Given k																
LDA	77	0.20	0.55	0.30	150	0.06	0.56	0.30	60	0.17	0.44	0.29	18	0.17	0.26	0.31
SBERT+k-means	77	0.55	0.82	0.65	150	0.21	0.83	0.66	60	0.39	0.71	0.52	18	0.42	0.63	0.56
BERTopic	77	0.56	0.83	0.64	150	0.26	0.82	0.65	60	0.45	0.73	0.58	18	0.48	0.64	0.61
LLM-embedding+k-means	77	0.62	0.85	0.72	150	0.28	0.87	0.70	60	0.40	0.73	0.51	18	0.49	0.70	0.62
ClusterLLM	77	0.60	0.84	0.70	150	0.21	0.85	0.68	60	0.47	0.77	0.59	18	0.44	0.68	0.59
Agentic clustering (ours)	77	0.70	0.88	0.79	150	0.28	0.88	0.72	60	0.62	0.81	0.69	18	0.71	0.78	0.78
Discover k																
BERTopic	90	0.60	0.84	0.69	166	0.26	0.82	0.70	67	0.43	0.73	0.57	67	0.27	0.64	0.36
TopicGPT	5	0.01	0.16	0.04	14	0.05	0.46	0.15	15	0.10	0.47	0.31	12	0.18	0.45	0.43
Huang & He	131	0.57	0.84	0.64	766	0.13	0.83	0.59	172	0.37	0.77	0.48	476	0.12	0.61	0.17
Agentic clustering (ours)	85	0.65	0.87	0.76	169	0.39	0.91	0.78	72	0.60	0.83	0.69	20	0.67	0.77	0.73

Method	GoEmotions				20 Newsgroups				StackExchange				Cost
	$\hat{k}$	ARI	NMI	ACC	$\hat{k}$	ARI	NMI	ACC	$\hat{k}$	ARI	NMI	ACC	total $
Given k													
LDA	27	0.01	0.04	0.10	20	0.19	0.36	0.34	121	0.01	0.31	0.08	$0
SBERT+k-means	27	0.02	0.07	0.11	20	0.42	0.58	0.61	121	0.16	0.52	0.29	$0
BERTopic	27	0.01	0.03	0.24	20	0.31	0.53	0.39	70	0.17	0.51	0.30	$0
LLM-embedding+k-means	27	0.04	0.13	0.17	20	0.49	0.63	0.64	121	0.34	0.67	0.47	$0.39
ClusterLLM	27	0.06	0.21	0.25	20	0.48	0.64	0.67	121	0.31	0.66	0.45	$1.53
Agentic clustering (ours)	27	0.08	0.24	0.35	20	0.56	0.68	0.74	121	0.44	0.73	0.56	$100 + $22
Discover k													
BERTopic	429	0.01	0.09	0.22	272	0.17	0.51	0.20	70	0.17	0.51	0.30	$0
TopicGPT	27	0.00	0.04	0.11	18	0.28	0.53	0.39	30	0.13	0.54	0.28	$100 + $66
Huang & He	207	0.03	0.17	0.16	377	0.22	0.53	0.29	2764	0.02	0.69	0.10	$100 + $44
Agentic clustering (ours)	27	0.08	0.24	0.32	20	0.56	0.68	0.74	130	0.42	0.73	0.56	$100 + $22

Table 2: Clustering results across seven benchmarks. Cost is the total USD across all seven datasets; methods that use the Claude Code Max subscription show it as the flat $100 subscription plus metered API spend ($100 + API).

the methods that can handle it natively, we use the no-none cluster mode when there is no gold none class, and the with-none cluster mode when there is a gold none class.

Document-length cap. For the LLM-using baselines, every call that passes a document body to a language model truncates the text to 512 tokens: ClusterLLM's triplet inputs, TopicGPT's per-document assignment, and Huang & He's label-generation and classification inputs. The cap is applied once, in each baseline's dataset adapter, so the 512-token text is that method's unit of analysis throughout. It binds mainly on 20 Newsgroups and the longest StackExchange posts; the five short-utterance datasets are unaffected.

Seeds and sampling. All methods are run with a single seed, owing to the cost of the full sweep.

Gold set leakage. For two methods, TopicGPT and Huang & He, the headline reported results are computed by using a sample of the gold set in the prompts. This helps inform the granularity of the clusters produced. However, this is tantamount to gold set leakage. Therefore we run these methods without prompting with the gold set (referred to as the '0% seed' configuration in the original paper).

Metrics. We report Adjusted Rand Index (ARI), Normalised Mutual Information (NMI), and clustering accuracy under Hungarian assignment (ACC).[3] The three metrics disagree in informative ways on datasets with a gold "none" class: NMI tolerates a method redistributing "none" documents across in-scope clusters; ARI and ACC penalise it.

6 Results

Table 2 reports the results of the experiments. We find that our method is state-of-the-art on all datasets, with the exception of CLINC150, where it draws with LLM-embedding+k-means on ARI. For many datasets, our method is more than 10% better than the second-best method on ARI.

[3] ARI and NMI use the standard scikit-learn implementations; ACC is computed via `scipy.optimize.linear_sum_assignment` on the predicted$\times$gold contingency matrix.

Other LLM taxonomy models struggle to control granularity. Without the gold-set injection, both Huang & He and TopicGPT struggle to control the granularity of the clusters produced. Huang & He produces a large number of small clusters, while TopicGPT produces a small number of large clusters, leading them to perform worse than LDA on some datasets. In both cases, their fixed prompts ask the LLM to "categorize" inputs but contain no signal about the abstraction level at which categories should be drawn — and because the surrounding pipeline is programmatic, there is no inner loop that could adapt that signal from the data. The MASSIVE-Intent / MASSIVE-Domain pair is the natural experiment that exposes this: the two datasets share the same 2,974 utterances and differ only in granularity and clustering intent, yet both TopicGPT and Huang & He are provided with the exact same prompt in both cases. This highlights the importance of user-given clustering intents, which is explored further in Section 6.

Dense embedding models. LLM-embedding+k-means beats LDA, SBERT+k-means, and BERTopic on every dataset on the majority of metrics, in line with the findings of Petukhova et al. (2025). ClusterLLM's triplet fine-tuning closes most of that gap, sometimes more.

Ablations. We ablate two components of our method in the discover-k setting (Table 3, Appendix C). First, we run our method without the optional user cluster intent. Removing this description significantly degrades performance on several of the datasets, highlighting the importance of clustering intent.

Second, we remove the post-synthesis refinement loop (auditor, critic, investigator). This has little effect, demonstrating that most of the power of this method comes from the sampling, prompting and synthesizing.

Cost. The final column of table 2 reports total dollar spend per method, summed across the seven datasets. Our method's figure follows one reporting convention. Its agent loop (Proposer, Synthesizer, Auditor, Critic, Investigator) is driven by Claude Code on a Claude Code Max subscription rather than the metered API, because that is how the method is deployed — there is no realistic API-only version of it. We therefore report this component as the literal subscription tier (∼$100 per month) rather than an API-equivalent token estimate, and add to it the metered cheap-tier (`gpt-5-mini`) spend for the full-corpus classification pass. For equivalence, we estimate the cost of Huang & He and TopicGPT as $100 for all API calls using the higher-tier model, plus the actual API call cost for usage of the lower-tier model.

The classical baselines (LDA, SBERT+k-means, BERTopic) run locally at no API cost, and the metered embedding baselines are inexpensive (LLM-embedding+k-means $0.39, ClusterLLM $1.53). The costliest methods are those that send a language model over the entire corpus: Huang & He ($44), whose label-generation and classification phases both scan every document, and our method ($22).

The comparison is consequently not perfectly like-for-like, but reflects the fairest comparison of costs. Notably, our method's quality lead does not come at the highest price.

7 Conclusion

We have presented an agentic approach to text clustering, in which an orchestrator dispatches a small set of specialised LLM agents — proposer, synthesizer, auditor, investigator, critic — in a state-dependent order rather than executing the fixed pipeline of prior LLM-driven taxonomy work. Our method is state-of-the-art or tied on all datasets, and for many datasets is more than 10% better than the second-best method on ARI.

Limitations

Re-implementation of baselines. Every method was re-implemented while holding the use of privileged information (such as gold-label seeding) and LLMs consistent across methods. We have tried to keep these re-implementations as faithful as possible — preserving each method's prompts, hyperparameters, and pipeline structure — and have documented every deviation in Appendix B. Even so, perfect comparability across methods built around very different LLM workflows is not achievable, and small cross-method differences in score may reflect implementation choices rather than the underlying methods.

Single seed. Owing to the cost of the full sweep — seven datasets across eight methods, several of which make per-document LLM calls — every reported number is a single-seed run. We do not characterise within-method variance, so small ranking swaps between methods should not be over-

interpreted; the gaps we draw conclusions from (typically more than 10% ARI) are well outside any plausible single-seed noise envelope.

English only. All seven benchmarks are English. We do not evaluate on non-English or code-mixed corpora and make no claim that the agentic pipeline's quality lead transfers to other languages.

Closed-model dependence. The method depends on frontier closed models (Claude Opus via Claude Code, GPT-5-mini via the OpenAI Batch API), which limits reproducibility, locks users to specific vendors, and means behaviour can shift silently as those models are updated. Open-weight substitution is possible in principle — the orchestrator and per-agent prompts are model-agnostic — but is unevaluated here.

Risks. Our method inherits three risks worth flagging. First, taxonomies discovered by the pipeline may be applied to consequential downstream decisions — routing support tickets, triaging moderation queues, categorising sensitive documents — and misclassifications in those settings carry material costs that are not captured by the aggregate metrics we report. Second, every agent in the loop is an LLM call, so whatever biases the underlying models carry — in what they treat as a natural category, what they label as noise, how they handle minority dialects or non-Western contexts — propagate into the resulting taxonomies, which is particularly relevant for corpora about people. Third, the user-supplied task description is a feature for legitimate analysis but also lowers the barrier to partitioning a corpus along any axis a user names (e.g. political affiliation or demographic markers), amplifying a dual-use concern that is not novel to this work but is made more accessible by it.

References

Pritom Saha Akash, Trisha Das, and Kevin Chen-Chuan Chang. 2024. TopicAdapt — an inter-corpora topics adaptation approach. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*.

Dimo Angelov. 2020. Top2Vec: Distributed representations of topics. *arXiv preprint arXiv:2008.09470*.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

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.

Maarten De Raedt, Frédéric Godin, Thomas Demeester, and Chris Develder. 2023. IDAS: Intent discovery with abstractive summarization. In *Proceedings of the 5th Workshop on NLP for Conversational AI (NLP4ConvAI 2023)*.

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. 2020. GoEmotions: A dataset of fine-grained emotions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4040–4054.

Zijin Feng, Luyang Lin, Lingzhi Wang, Hong Cheng, and Kam-Fai Wong. 2024. LLMEdgeRefine: Enhancing text clustering with LLM-based boundary point refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18455–18462.

Jack FitzGerald, Christopher Hench, Charith Peris, Scott Mackie, Kay Rottmann, Ana Sanchez, Aaron Nash, Liam Urbach, Vishesh Kakarala, Richa Singh, Swetha Ranganath, Laurie Crist, Misha Britan, Wouter Leeuwis, Gokhan Tur, and Prem Natarajan. 2023. MASSIVE: A 1M-example multilingual natural language understanding dataset with 51 typologically-diverse languages. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4277–4302.

Maarten Grootendorst. 2022. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794*.

Chen Huang and Guoxiu He. 2024. Text clustering as classification with LLMs. *Preprint*, arXiv:2410.00927. Subsequently SIGIR-AP 2025. Repo: ECNU-Text-Computing/Text-Clustering-via-LLM.

Jagadeesh Jagarlamudi, Hal Daumé III, and Raghavendra Udupa. 2012. Incorporating lexical priors into topic models. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 204–213.

Ken Lang. 1995. NewsWeeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pages 331–339.

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*, pages 1311–1316.

Alina Petukhova, João P. Matos-Carvalho, and Nuno Fachada. 2025. Text clustering with large language model embeddings. *International Journal of Cognitive Computing in Engineering*, 6.

Chau Minh Pham, Alexander Hoyle, Simeng Sun, Philip Resnik, and Mohit Iyyer. 2024. TopicGPT: A prompt-based topic modeling framework. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*.

Vijay Viswanathan, Kiril Gashteovski, Carolin Lawrence, Tongshuang Wu, and Graham Neubig. 2024. Large language models enable few-shot clustering. *Transactions of the Association for Computational Linguistics*, 12:321–333.

Mengting Wan, Tara Safavi, Sujay Kumar Jauhar, Yujin Kim, Scott Counts, Jennifer Neville, Siddharth Suri, Chirag Shah, Ryen W. White, Longqi Yang, Reid Andersen, Georg Buscher, Dhruv Joshi, and Nagu Rangan. 2024. TnT-LLM: Text mining at scale with large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Han Wang, Nirmalendu Prakash, Nguyen Khoi Hoang, Ming Shan Hee, Usman Naseem, and Roy Ka-Wei Lee. 2023. Prompting large language models for topic modeling. In *Proceedings of the 2023 IEEE International Conference on Big Data (BigData)*, pages 1236–1241.

Yuwei Zhang, Zihan Wang, and Jingbo Shang. 2023. ClusterLLM: Large language models as a guide for text clustering. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13903–13920.

A Dataset processing details

All loaders are deterministic given a fixed scikit-learn version (for 20 Newsgroups) and the Hugging Face dataset revisions recorded in each dataset's `meta.json`.

Each dataset is used for the intent classification, emotion classification, or topic clustering purpose for which it was originally released, which aligns with our text-clustering evaluation; per-dataset license terms are documented at the end of each subsection below.

A.1 Banking77

Banking77 (Casanueva et al., 2020) consists of customer-service queries to an online bank, labelled with one of 77 fine-grained intents (e.g. *activate_my_card*, *pending_top_up*). We use the 3,080-example test split unmodified, matching prior LLM-clustering work. The dataset is loaded from `PolyAI/banking77` on Hugging Face Hub via the `refs/ convert/parquet` revision. The `ClassLabel` feature provides the canonical label names, which we copy directly into the taxonomy. *License:* CC-BY-4.0, consistent with research re-use.

A.2 CLINC150

CLINC150 (Larson et al., 2019) is a 150-intent benchmark with a dedicated out-of-scope (OOS) class for queries outside the domain. We use the `plus` config's 5,500-example test split (4,500 in-scope + 1,000 OOS). Unlike some prior LLM-clustering work, which discards OOS examples, we retain them and evaluate them as the gold "none" class; methods without native "none" support are expected to score worse here. The dataset is loaded from `clinc/clinc_oos` on Hugging Face Hub. The dataset's parquet conversion does not preserve config names as Hugging Face dataset configs; instead, each config lives as a subdirectory of the `refs/convert/parquet` branch. We load directly from `hf://datasets/clinc/clinc_oos@refs/convert/parquet/plus/test/0000.parquet`. Examples with source intent "oos" are remapped to `gold_label_id` $= -1$ and `gold_label_name` = `__none__`; the remaining 150 in-scope intent ids are compacted to a contiguous $0 \ldots 149$ range. Two prior LLM-clustering papers use different CLINC configurations and both drop the OOS examples: Zhang et al. (2023) use the same `plus` test split as we do but discard OOS, while Viswanathan et al. (2024) use the `small` configuration. We use `plus` so that the OOS subset is available, and retain it as the gold "none" class. *License:* CC-BY-3.0, consistent with research re-use.

A.3 MASSIVE-Intent and MASSIVE-Domain

MASSIVE-Intent and MASSIVE-Domain (FitzGerald et al., 2023) share the same 2,974 EN test utterances under two different label projections: 60 fine-grained intents and 18 coarse scenarios respectively. This pair isolates the effect of taxonomy

granularity from corpus content. The two splits are loaded from `mteb/amazon_massive_intent` and `mteb/amazon_massive_scenario` respectively, both using the `en` configuration and the `test` split. In both datasets the gold label ships as a human-readable string (e.g. *alarm_set*, *calendar*); we assign integer ids in sorted-string order to obtain a stable mapping. To keep the gold label space exactly k=60 for MASSIVE-Intent (per the MASSIVE paper) even though the test split is missing one of the 60 intents (*cooking_query*), the canonical label set is derived from the union of train, validation, and test labels — only test rows are then emitted as documents. MASSIVE-Domain uses an analogous union to fix k=18. The processing pipeline cross-checks that the EN test texts in MASSIVE-Intent and MASSIVE-Domain coincide row-by-row, since the two MTEB datasets are different label projections of the same corpus and any drift would silently break the comparison. *License:* CC-BY-4.0, consistent with research re-use.

A.4 GoEmotions

GoEmotions (Demszky et al., 2020) comprises Reddit comments labelled with 27 emotions plus a neutral class. Like with CLINC's OOS, we retain neutral as the gold "none" class, departing from prior work that drops it (Zhang et al., 2023). The dataset is loaded from `google-research-datasets/go_emotions` on Hugging Face Hub, using the `simplified` configuration. The `simplified` config restricts to single-label rows but ships them as a list (`labels: list[int]` of length one); rows whose list length is not one are skipped (8,817 multi-label rows). The remaining 45,446 single-label documents are concatenated across train, validation, and test — the same recipe used by Zhang et al. (2023). *Neutral* (source label id 27) is remapped to `gold_label_id` $= -1$ and `gold_label_name` = `__none__`, and the remaining 27 in-scope emotion ids are compacted to $0 \ldots 26$. *License:* Apache-2.0, consistent with research re-use.

A.5 20 Newsgroups

20 Newsgroups (Lang, 1995) is a long-form topic-clustering benchmark of Usenet posts across 20 topics. We adopt the BERTopic preprocessing recipe (Grootendorst, 2022) via scikit-learn (version 1.8.0) using `fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))`. The `remove` flags strip the header block, the signature/footer block, and quoted reply text — all three contain newsgroup names or quote markers that would otherwise leak the gold label. After stripping we drop rows that become empty whitespace (515 documents), leaving 18,331 documents across the 20 newsgroup labels. We use scikit-learn rather than `SetFit/20_newsgroups` on Hugging Face Hub because the latter does not expose the `remove` parameter. We do not use MTEB's `twentynewsgroups-clustering`, which is subject-line only ($\sim$32 chars on average) and would not exercise the long-document regime that motivates this dataset's inclusion. *License:* the 20 Newsgroups corpus is in the public domain and has been freely redistributed for research for three decades, including by scikit-learn under its BSD-3-Clause distribution; our use is consistent with this.

A.6 StackExchange

StackExchange consists of 4,156 forum question titles drawn from 121 stackexchange.com sites, where the site identity (e.g. *english*, *history*) serves as the gold cluster label. We use the `small` subset released by Zhang et al. (2023), loaded from the data zip linked in the `ClusterLLM` GitHub repository's README. The loader auto-downloads the zip to `data/raw/clusterllm/` on first run and reads `datasets/stackexchange/small.jsonl`. We strip the `.txt` suffix that ClusterLLM's recipe appends to the site names, yielding 121 in-scope labels. We adopt this resource rather than Hugging Face's `mteb/stackexchange-clustering` dataset because the latter is structured as a list of 25 clustering subsets (5,010–26,666 documents each, 121 distinct labels in total), and MTEB's reported score for this task is the mean V-measure across per-subset k-means runs. That protocol is incompatible with our single-partition ARI/NMI/ACC reporting on a fixed taxonomy. The ClusterLLM-small file is the de facto small-StackExchange benchmark in the LLM-clustering line of work, and at n=4,156 sits inside the size range of our other six datasets. *License:* the underlying Stack Exchange user content is CC-BY-SA 4.0 (Stack Exchange's site-wide content licence); the ClusterLLM repackaging is released alongside the paper without an explicit licence for the bundle, and we use it under the same research-reproduction terms applied by prior work (Zhang et al., 2023).

B Implementation details

Compute and infrastructure. The dominant compute cost in our pipeline and in the four LLM-using baselines (LLM-embedding+k-means, ClusterLLM, TopicGPT, and Huang & He) is consumed by closed-model API calls (Claude Opus 4.7, GPT-5-mini, and OpenAI `text-embedding-3-large`); we account for this as USD spend in the Cost paragraph of §6 and the final column of Table 2 rather than in GPU hours. Local compute is light: LDA, the SBERT-based pipelines (SBERT+k-means, BERTopic, and ClusterLLM's Instructor encoder at inference), and the per-dataset processing all run on a single workstation in minutes per dataset. The only non-workstation step is ClusterLLM's perspective fine-tuning (the upstream `finetune.sh`), which runs on a single NVIDIA A100 for approximately ten minutes per dataset. The open-weight encoders used in this study are `sentence-transformers/all-mpnet-base-v2` (110M parameters) and `hkunlp/instructor-large` (335M parameters); the closed models we call (Claude Opus 4.7, GPT-5-mini, `text-embedding-3-large`) do not publish parameter counts.

LDA. scikit-learn's `LatentDirichletAllocation` (batch variational inference, 20 iterations) with K set to the in-scope class count of the dataset, fit on a `CountVectorizer` feature set with the standard English stop-word list, `min_df=2` and `max_df=0.95`. The per-document cluster assignment is the argmax of the document–topic distribution; the per-topic top words from ϕ are saved as cluster descriptions but not used by any metric. *Licences:* scikit-learn is BSD-3-Clause.

SBERT+k-means. `sentence-transformers/all-mpnet-base-v2` sentence embeddings (768-dim), L2-normalised, fed to scikit-learn's k-means with `n_init=10` and a maximum of 300 iterations. L2-normalisation turns Euclidean k-means into cosine k-means on the unit sphere, the standard convention for SBERT embeddings. Embeddings are cached on disk so that subsequent methods using the same encoder (e.g. BERTopic) reuse them rather than recomputing. *Licences:* `all-mpnet-base-v2` is Apache-2.0; the `sentence-transformers` library is Apache-2.0; scikit-learn is BSD-3-Clause.

BERTopic. The standard SBERT $\rightarrow$ UMAP $\rightarrow$ HDBSCAN $\rightarrow$ class-based TF-IDF pipeline of Grootendorst (2022), using the same `all-mpnet-base-v2` embeddings as SBERT+k-means (the cache is shared). UMAP is configured with `n_neighbors=15`, `n_components=5`, `min_dist=0.0`, cosine metric, and an explicit `random_state` for reproducibility; HDBSCAN with `min_cluster_size=10` and the default EOM cluster-selection method; c-TF-IDF over a `CountVectorizer` with the standard English stop-word list. We report two configurations per dataset: a given-k run with `nr_topics` set to the in-scope class count, which uses BERTopic's post-hoc hierarchical merging step to reduce HDBSCAN's discovered topics to the requested number, and a discover-k run with `nr_topics=None` that keeps HDBSCAN's natural output. Per the noise-handling rule of §5, we call `reduce_outliers(strategy='embeddings')` on the five datasets without a gold "none" class to force-assign every document, and skip it on CLINC and GoEmotions so that HDBSCAN's noise cluster (-1) is retained as BERTopic's native "no cluster" output and aligned to the gold -1 class. *Licences:* BERTopic, UMAP, and HDBSCAN are all MIT-licensed.

LLM-embedding+k-means. OpenAI `text-embedding-3-large` embeddings (3072-dim), obtained through the OpenAI Batch API (50% discount, $\leq$24h SLA) per §5, then fed to the same scikit-learn k-means clusterer as SBERT+k-means (`n_init=10`, 300 iterations maximum, L2-normalised, K set to the in-scope class count). Document inputs are truncated to the encoder's 8,192-token ceiling before embedding. The resulting vectors are cached on disk together with the actual token usage and USD, so the cost column reports what we paid rather than a re-estimate. This is the LLM-embedding pipeline reported by Petukhova et al. (2025), instantiated with a current frontier embedder. *Licences:* OpenAI `text-embedding-3-large` is a proprietary commercial service used under the OpenAI API Terms of Use; scikit-learn is BSD-3-Clause.

ClusterLLM. We run the triplet ("perspective") variant of ClusterLLM (Zhang et al., 2023), with given-k, from the authors' code with its default hyperparameters throughout (Instructor-large embeddings, `hkunlp/instructor-large`; 1,024 sampled triplets; the upstream `scripts/finetune.sh`

	Full (discover-k)			− refinement loop			− task description		
Dataset	ARI	NMI	ACC	ARI	NMI	ACC	ARI	NMI	ACC
Banking77	0.65	0.87	0.76	0.66	0.87	0.76	0.66	0.88	0.75
CLINC150	0.39	0.91	0.78	0.44	0.91	0.80	0.24	0.88	0.74
MASSIVE-Intent	0.60	0.83	0.69	0.60	0.82	0.69	0.57	0.82	0.69
MASSIVE-Domain	0.67	0.77	0.73	0.67	0.77	0.74	0.56	0.74	0.65
GoEmotions	0.08	0.24	0.32	0.07	0.24	0.32	0.07	0.20	0.26
20 Newsgroups	0.56	0.68	0.74	0.56	0.68	0.75	0.47	0.66	0.68
StackExchange	0.42	0.73	0.56	0.41	0.72	0.56	0.43	0.73	0.56

Table 3: Ablations on our method (discover-k).

fine-tuning settings). We make two changes. First, triplet judging is run with `gpt-5-mini` via the OpenAI Batch API. Second, we drive Instructor through `sentence-transformers` 5.x's `prompt=` API rather than the authors' `sentence-transformers` 2.2 wrapper, which no longer runs under current libraries; weights and instructions are unchanged. *Licences:* the upstream ClusterLLM code is released without a LICENCE file and is vendored here under a research-reproduction rationale; `hkunlp/instructor-large` is Apache-2.0; `gpt-5-mini` is a proprietary OpenAI service used under the OpenAI API Terms of Use.

TopicGPT. We run TopicGPT (Pham et al., 2024) from the authors' code at recursion depth 1, producing a flat taxonomy that matches our gold-label format. Topic generation and refinement run on the frontier tier (`claude-opus-4-7` via the Claude Code subscription); per-document assignment and the follow-up correction pass run on the cheap tier (`gpt-5-mini` via the OpenAI Batch API). For topic generation, we provide no seed topics, so the taxonomy is discovered from the corpus alone — matching the fully-unsupervised, no-privileged-information setting we apply to every LLM baseline — and we use the paper's recommended `early_stop`=200. We add a defensive cap of 1,000 documents scanned for datasets where the model keeps discovering new topics and never triggers early stop (GoEmotions and StackExchange), as it would otherwise run through the whole corpus. *Licences:* the upstream TopicGPT code is released without a LICENCE file and is vendored here under a research-reproduction rationale; Claude Opus and `gpt-5-mini` are proprietary services used under the Anthropic and OpenAI API Terms of Use respectively.

Huang & He. We run Huang and He (2024) from the authors' code with its default hyperparameters (label generation over chunks of B=15 documents; the generate → merge → classify pipeline with no outer iteration). We run their 0%-seed configuration — passing an empty `given_labels` list into the verbatim upstream label-generation prompt — rather than their 20%-seeded headline, which injects gold-label names and would give the method privileged information no other baseline receives. Label generation and per-document classification run with `gpt-5-mini` via the OpenAI Batch API; the single label-merging call per dataset runs on the frontier tier (`claude-opus-4-7` via the Claude Code subscription). *Licences:* the upstream Huang & He code is released without a LICENCE file (the upstream GitHub API reports `"license": null`) and is vendored here under a research-reproduction rationale; Claude Opus and `gpt-5-mini` are proprietary services used under the Anthropic and OpenAI API Terms of Use respectively.

Our method. For all benchmark runs we use the default classification prompt, built directly from the finalised taxonomy as described in §3.8, and do not run the optional prompt-tuning step. The only labels available on the benchmark datasets are the gold partition itself, so tuning the classifier against them would leak the evaluation signal into the method under test. *Licences:* our agentic-clustering implementation is released under the MIT License; the underlying Claude Opus and `gpt-5-mini` models are proprietary services used under the Anthropic and OpenAI API Terms of Use respectively.

C Ablation results

Both ablations use the discover-k configuration. The − *refinement loop* row reuses each run's first synthesizer output, recovered from the run workspace, and re-runs only the cheap-tier classification against it; no frontier calls are incurred. The − *task description* row is a full end-to-end run with

`config.instructions` left empty — the plugin's documented "discover from the data alone" mode — and the dataset name removed from the orchestrator prompt, with the k-range and every other setting held identical to the main discover-k run, so the task description is the only changed variable.