Task Cascades for Efficient Unstructured Data Processing

Shreya Shankar
UC Berkeley
shreyashankar@berkeley.edu

Sepanta Zeighami
UC Berkeley
zeighami@berkeley.edu

Aditya G. Parameswaran
UC Berkeley
adityagp@berkeley.edu

ABSTRACT

Modern database systems increasingly allow users to query or process unstructured text or document columns—such as emails, reviews, or clinical notes—using LLM-powered functions. Users can express an operation in natural language (e.g., *"identify if this review complains about billing issues"*), with the system executing the operation on each document, in a row-by-row fashion. One way to reduce cost on a batch of documents is to employ the model cascade framework: a cheap proxy model processes each document, and only uncertain cases are escalated to a more accurate, expensive oracle model. However, model cascades miss out on important optimization opportunities; for example, often only part of a document is needed to answer a query, or other related, but simpler operations (e.g., *"is the review sentiment negative?", "does the review mention money?"*) can be handled by cheap models more effectively than the original operation, while still being correlated with it.

We introduce the ***task cascades*** framework, which generalizes model cascades by varying not just the model, but also the *document portion* and *operation* at each stage. Our framework uses an LLM agent to generate simplified, decomposed, or otherwise related operations and selects the most relevant document portions for each, constructing hundreds of candidate tasks from which it assembles a task cascade. We show that selecting an optimal cascade from a set of candidates is intractable through a reduction from Minimum Sum Set Cover; however, our iterative approach is able to construct an effective cascade. We also provide an extension that offers statistical accuracy guarantees: the resulting cascade meets a user-defined accuracy target (with respect to the oracle) up to a bounded failure probability. Across eight real-world document processing tasks at a 90% target accuracy, task cascades reduce inference cost by an average of nearly 50% compared to model cascades.

ACM Reference Format:
Shreya Shankar, Sepanta Zeighami, and Aditya G. Parameswaran. 2026. Task Cascades for Efficient Unstructured Data Processing. In *Under Revision for SIGMOD 2026.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Large language models (LLMs) are increasingly being integrated as operators in data management systems, enabling users to analyze unstructured text columns more easily. This line of work spans SQL extensions to databases, providing LLM-powered *map* and *filter* operations [16, 21, 55, 67], as well as Python-based declarative frameworks for executing pipelines of LLM-powered operators [1, 41, 49, 53, 65]. In all cases, users describe an operation in natural language, with the system executing it by invoking an LLM on each text value—hereafter referred to as a *document*—in a row-by-row fashion. Consider the following example:

Example 1.1 (Supreme Court Opinion Analysis). Suppose a legal analyst is working with a large collection of Supreme Court opinions. The analyst's goal is to determine, for each opinion, whether it overturns a lower court decision. In this setting, each *document* is a long, complex legal text, and the analyst poses an *operation* in natural language: *"Does this opinion overturn a lower court decision?"*

State-of-the-art LLMs like GPT-4o and GPT-4.1 can achieve high accuracy on tasks such as Example 1.1. However, commercial LLM APIs charge by the number of input and output tokens, making large-scale analysis costly. As a result, users look for ways to cut inference costs without losing much of the accuracy provided by the best models. They are often willing to accept a modest drop in accuracy—e.g., targeting 90% of GPT-4o's accuracy—in exchange for significant savings [49, 52]. Although cheaper models exist (such as GPT-4o-mini, which is nearly 17× cheaper than GPT-4o), they often deliver much lower accuracy.

Prior Work and Limitations. The standard approach to reduce costs of map and filter operations with machine learning models, while preserving output accuracy, is to use a *model cascade* [2, 34, 36, 37, 51]; with recent systems all adopting it as-is for LLM-powered operators [9, 49, 70]. Here, the system first applies a cheaper *proxy* model to each input document. If the proxy's confidence in its prediction exceeds a system-selected confidence threshold, the system accepts the proxy's output. Otherwise, the system escalates the input to a more accurate but more expensive *oracle* model (e.g., the best available LLM). Users specify a performance target (e.g., 90% of the oracle's accuracy), and the system tunes the confidence thresholds to meet this target while minimizing cost.

However, the model cascades framework, by focusing only on swapping models, underutilizes optimization opportunities. For instance, when no cheaper model can perform the operation accurately, documents will end up being processed by the oracle. In fact, on medical or legal tasks, cheaper models may have as little as 30% of the accuracy of expensive models [17, 47, 63], with the gap being even wider for long-context reasoning [42, 68]. Moreover, model cascades always process the full document, even though many operations require only a small, relevant portion of the text.

Change the Task, Not Just the Model. Our key insight is that *effective cascades should vary the operation as well as the portion of the document*, in addition to the model. Earlier stages of the cascade can execute any operation that is easier for cheaper models—such as a prerequisite check, a decomposed sub-task, or any related operation that is correlated with the original. We refer to such

SIGMOD, June 2026, Bengaluru, India

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

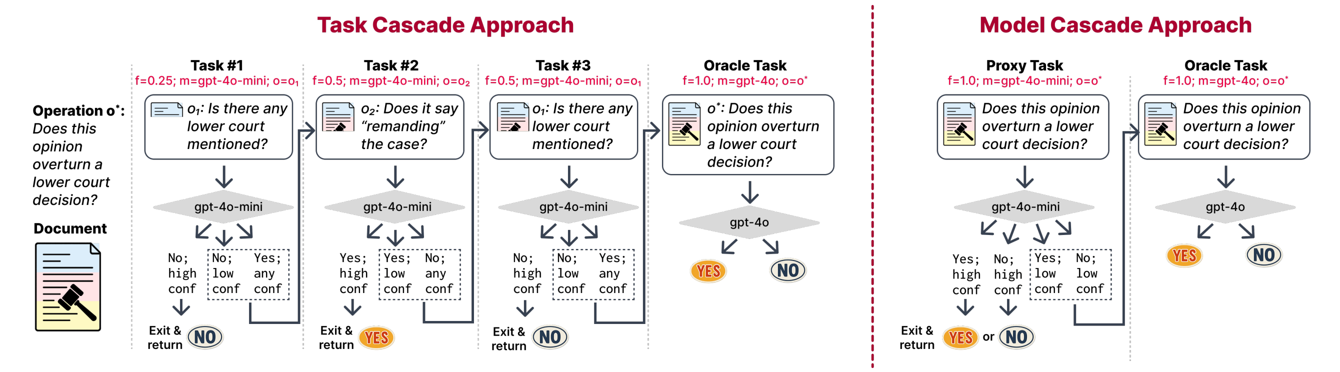

Figure 1: Task vs. model cascade for determining whether a Supreme Court opinion overturns a lower court decision in Example 1.1. In a task cascade (left), each stage (or task) is defined by a document fraction (f), a model (m), and an operation (o). Surrogate operations may be reused at different document fractions (as in tasks 1 and 3). The final oracle task applies the original user-specified operation (o^*) on the full document ($f = 1.0$) with the oracle model (m =gpt-4o) if prior tasks cannot confidently resolve the input. In contrast, a model cascade (right) applies the same operation o^* to the full document at each stage, varying only the model.

operations as *surrogate* operations: variants of the original operation that are easier for a proxy model to perform. In Example 1.1, rather than having a proxy model decide if an opinion overturns a lower court decision, we can have it check whether the opinion mentions a lower court at all. If not, we don't need to invoke the oracle.

Another complementary strategy to improve proxy model accuracy is to *prune irrelevant context* from each document before inference. For example, instead of providing the entire document to the LLM, we can supply only the sections that are semantically relevant to the operation—such as paragraphs containing "overturn," "judgment vacated," or their synonyms. By eliminating unrelated text, the proxy model is more likely to perform the operation both correctly *and* with lower cost [58, 69].

In this paper, we introduce the notion of a **task cascade**: an ordered sequence of *tasks*, each task defined by an operation, an LLM, and a selected fraction of the document to analyze. As illustrated in Figure 1, for Example 1.1, a task cascade may first check if there is any lower court mentioned in a short excerpt before switching over to a different surrogate, and later revisit this same operation over even more of the document—before finally applying the original operation with the oracle model on the full document if needed.

Task Cascade Challenges. However, designing an effective task cascade is challenging for several reasons. First, discovering effective surrogate operations is hard, and there is an infinitely large search space. While one can use an LLM to author a new operation, a surrogate operation is only beneficial if it is both accurately performed by a cheap model *and* if it resolves a substantial fraction of documents. A poor surrogate can potentially make the entire cascade *less* efficient than if it were omitted. Second, determining the ideal portion of the document for each task is non-trivial. The goal is to minimize token costs by providing just enough context for an accurate decision; too little text risks errors, while too much (like a full document) negates potential savings and can overwhelm cheaper models. Third, determining the best order in which to apply tasks is difficult. The number of possible sequences grows rapidly with more tasks, and each sequence can differ significantly in cost and effectiveness. A task may perform relatively well overall but poorly when applied only to the harder examples that earlier tasks did not resolve, and thus not meet accuracy constraints. Moreover, LLM APIs offer cost savings (50–90%) when multiple prompts share the same input prefix. As a result, optimizing task order is not just about accuracy and selectivity of tasks: it also involves structuring inputs to exploit reuse. ***Overall, jointly optimizing surrogate operations, document fractions, and task orders creates a complex search space that is difficult to reliably navigate.***

Effective Task Cascades. We introduce an approach for constructing effective task cascades, given a document set, user-specified operation, and a target accuracy with respect to an oracle. We address the aforementioned challenges as follows:

- **Task ordering.** Given a set of tasks, trying all orderings to find the best one quickly becomes infeasible as the number of tasks grows. We show that optimally ordering tasks is NP-Hard through a reduction from Minimum-Sum-Set-Cover, and inspired by approximation algorithms for this problem, we develop a greedy algorithm that sequentially adds the task that most reduces total inference cost while satisfying accuracy constraints.
- **Confidence threshold selection.** Per task in the cascade, we need to set confidence thresholds that determine whether to accept the proxy model output for that task. A naive approach sets thresholds for each task so that, by a union bound, the overall cascade meets the target accuracy with the desired probability. However, as the number of tasks increases, the naive method becomes overly conservative, making it difficult to find a feasible cascade. We develop a new threshold adjustment algorithm, building on recent statistical results [66], which enables us to achieve accuracy guarantees with far fewer samples.
- **Surrogate operation generation.** To generate the set of tasks in the first place, a simple idea is to prompt an LLM agent (e.g., based on OpenAI's o1) for candidate operations, but these surrogates are often not effective for smaller models, as the agent has no way to know which operations are actually easy for cheaper models to perform. Instead, we introduce an *iterative* approach, where the agent proposes surrogates, and tests and refines them based on which are actually performed well by proxies.
- **Document pruning.** Finally, per task, we want to minimize the portion of the document provided as input, while maintaining

accuracy. A naive approach to pruning irrelevant portions might use embedding similarity to select document chunks most related to the user operation, similar to standard retrieval-augmented generation (RAG) techniques; however, this approach often fails for complex or long-context tasks, so we instead train a lightweight classifier—supervised by the oracle LLM—to score each chunk's relevance and use these scores for pruning.

Overall, our contributions include:

(1) We introduce the concept of *task cascades* for LLM-based document processing, generalizing model cascades by allowing each stage to specify a model, operation (original or surrogate), and document fraction.

(2) We show that the problem of optimal cascade construction is NP-Hard, via a reduction from Minimum Sum Set Cover.

(3) We develop an automated, agentic approach for constructing effective task cascades—jointly discovering surrogate operations, pruning documents, and greedily assembling task sequences that minimize cost while meeting accuracy targets. We show that our approach can be extended to provide guarantees.

(4) We evaluate our method on eight complex document classification tasks drawn from Kaggle and prior LLM-based data processing research, comparing against standard model cascade baselines and performing comprehensive ablations to assess the contribution of each component of our approach (i.e., task ordering, agentic surrogate generation, document pruning). Across all workloads at a 90% target accuracy, ***our approach reduces inference cost by 48.5% on average compared to model cascade baselines, and by 86.2% compared to using the oracle model alone***.

We present the problem and an overview of our approach in Section 2. Next, in Section 3, we show the hardness of optimal task ordering and describe our greedy cascade assembly algorithm and new procedure for achieving statistical accuracy guarantees. In Section 4, we describe how we prune irrelevant context in documents, followed by our agentic approach for discovering surrogate operations in Section 5. Then, in Section 6, we describe how to model the cost of task cascades. Finally, we present our evaluation in Section 7 and cover related work in Section 8.

2 PROBLEM SETUP AND APPROACH

In this section, we formalize the notion of task cascades and give an overview of our approach to finding an efficient task cascade. A summary of all notation is provided in Table 1, for convenience.

2.1 Setup

Problem Setting. We focus on the setting where LLM-powered *map* and *filter* operations in commercial databases [16, 21, 55, 67] or recent systems [41, 49, 53] produce outputs from a fixed set of classes. Formally, the user provides a collection D of documents and a target operation o_{orig}, described in natural language. The goal is to assign each document $x \in D$ to a class c from a predefined set of classes C. The user specifies an accuracy target $\alpha \in (0, 1]$, requiring the system's predictions to match an oracle model m_{oracle}'s predictions on at least an α-fraction of documents.

Given a document x (or, alternatively, a fraction of the document, as described below) and operation o (either the original or a surrogate, defined below), a model m produces a score $p_m(c \mid x, o)$ for each class $c \in C$. The class c with the highest $p_m(c \mid x, o)$ is taken as the prediction of the model. For Example 1.1, x could be a court opinion, o might be *"Does this opinion overturn a lower court decision?"*, and $c \in \{\text{True}, \text{False}\}$. To enable early termination in a cascade, we use **confidence thresholds**: for each class $c \in C$, if the model's confidence $p_m(c \mid x, o) \geq \tau^c$ for some threshold τ^c, the prediction for that document x is accepted; otherwise, x continues to the next stage.

Instead of always using the full document x, we may also use a portion of the document, x_f, denoting the top f fraction of x, containing the most relevant content for performing o_{orig}. Relevance is determined by a scoring function that ranks document segments by their utility for o_{orig}. So, the confidence threshold now based on $p_m(c \mid x_f, o)$. A **task** is then defined as $T_i = (m_i, o_i, f_i, \tau_i)$, where:

- m_i: model (e.g., proxy or oracle)
- o_i: operation (original or surrogate)
- f_i: fraction of the document processed
- $\tau_i = \{\tau_i^c\}_{c \in C}$: class-specific confidence thresholds

A **task cascade** is an ordered sequence of tasks $\pi = (T_1, T_2, \ldots, T_k)$. At inference time, for a given document x, we evaluate each T_i in sequence: model m_i is applied to input x_{f_i} with operation o_i, yielding a predicted class c and confidence score $p_{m_i}(c \mid x_{f_i}, o_i)$. If this confidence exceeds the corresponding threshold τ_i^c, the prediction is accepted, $\text{Cascade}(\pi, x) = c$, and the cascade terminates; otherwise, processing continues to the next task. If none of the k tasks return a confident prediction, the cascade defers to the oracle task $T_{k+1} = (m_{\text{oracle}}, o_{\text{orig}}, 1, \varnothing)$, which applies o_{orig} to the full document with the oracle, with $\text{Cascade}(\pi, x)$ set to the oracle's prediction. Thus, a document "leaves" the cascade at the first confident prediction within π, or at the oracle if all tasks defer. A traditional model cascade is thus a "restricted" task cascade where each T_i uses the same operation $o_i = o_{\text{orig}}$ and processes the entire document ($f_i = 1.0$) at each stage, varying only the model m_i.

Confidence Scores. Each model m_i defines a distribution over output classes $c \in C$, conditioned on operation o_i and document portion x_{f_i}, i.e., $p_{m_i}(c \mid x_{f_i}, o_i)$. Inputs to LLMs are tokenized; a *token* is a subword unit such as a word fragment or punctuation mark. LLMs return log-probabilities for each token in the generated output, which can be transformed into confidence scores.

Cost Model. Each task $T_i = (m_i, o_i, f_i, \tau_i)$ sends a prompt to model m_i by concatenating the document fraction x_{f_i} and the operation o_i. We denote the number of tokens or *size* as $|\cdot|$, so $|x_{f_i}|$ is the size of the fractional document, and $|o_i|$ is the size of the operation prompt. We let λ_{in} and λ_{cached} denote the cost per new input token and per cached input token, respectively. For classification tasks, the output cost is typically insignificant and can be ignored, but can be easily incorporated if needed.

The cost to run task T_i on x_{f_i} depends on how much of the input is already cached from previous calls to the same model (as supported by LLM APIs). The cost is:

$$\text{Cost}(T_i, x) = \begin{cases} |x_{f_i}|\,\lambda_{\text{in}} + |o_i|\,\lambda_{\text{in}} & \text{if } x_{f_i} \text{ is new} \\ |x_{f_j}|\,\lambda_{\text{cached}} + (|x_{f_i}| - |x_{f_j}|)\,\lambda_{\text{in}} + |o_i|\,\lambda_{\text{in}} & \text{if } x_{f_j} \subseteq x_{f_i} \end{cases}$$

where x_{f_j} is the largest previously processed document prefix that is a subset of x_{f_i}. Throughout, we place the document before the

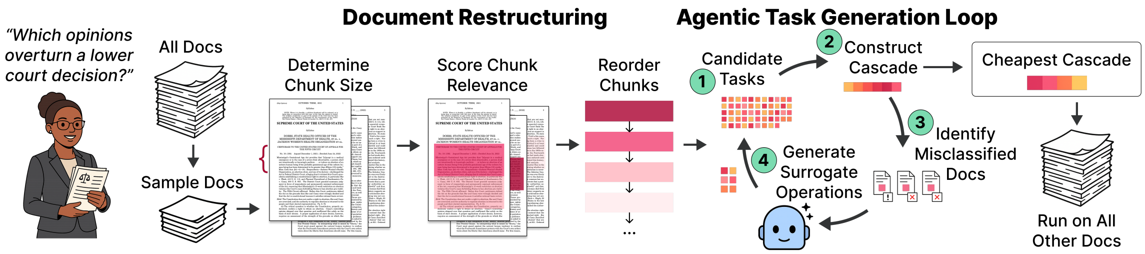

Figure 2: Overview of our approach. A user poses a complex classification query over long documents. Our approach restructures each document to prioritize relevant chunks, then iteratively proposes surrogate operations and assembles a cost-effective task cascade to meet accuracy constraints.

operation in the prompt, maximizing cache utilization of document tokens across multiple tasks using the same model. The total inference cost for document x is then:

$$\text{Cost}(\pi, x) = \sum_{i=1}^{j^*} \text{Cost}(T_i, x)$$

where j^* is the index of the first task to return a confident prediction (i.e., where $\text{Cascade}(\pi, x)$ is determined), or $j^* = k+1$ if the oracle is called on the original document.

Problem Statement. Given a document collection D, an original operation o_{orig}, a set of classes C, an oracle model m_{oracle}, available proxy models M, and an accuracy target α, we seek to construct a minimal-cost task cascade $\pi = (T_1, T_2, \ldots, T_k)$ as follows:

$$\min_{\pi} \quad \sum_{x \in D} \text{Cost}(\pi, x)$$

$$\text{s.t.} \quad \Pr\left[\frac{1}{|D|} \sum_{x \in D} \mathbb{I}\big[\text{Cascade}(\pi, x) = T_{k+1}(x)\big] \geq \alpha\right] \geq 1 - \delta$$

where $\mathbb{I}[\cdot]$ is the indicator function and T_{k+1} is the oracle task; i.e., $T_{k+1} = (m_{\text{oracle}}, o_{\text{orig}}, 1, \varnothing)$; so $T_{k+1}(x)$ denotes the oracle's prediction for x. $\delta \in (0, 1)$ is a user-specified upper bound on the probability that the cascade's accuracy on D falls below α.

While we focus on classification with fixed output classes, the task cascade framework can be extended to open-ended map or generation operations, provided there is a reliable automatic evaluator for correctness of an output (i.e., an LLM-as-a-judge [71] and a means to compute confidence scores, e.g., the geometric mean of per-token probabilities as in [24].

2.2 Overview of Our Approach

Our approach constructs a cost-effective task cascade π in an offline phase, using a small, representative sample of the document collection D, which we'll refer to as D_{dev}, the *development set*. The complete procedure is summarized in Algorithm 1. The construction involves: (i) restructuring documents to support efficient fractional processing, and (ii) an agentic loop that iteratively refines a set of candidate tasks and assembles an optimized cascade. We focus on a two-model setting—a cheap proxy model (e.g., GPT-4o-mini) and a more accurate, expensive oracle (e.g., GPT-4o)—as in model cascades, but our approach can generalize to any number of models. An overview of our approach is depicted in Figure 2.

2.2.1 Document Restructuring. We pre-process each document to ensure that the most relevant content for the original operation

Algorithm 1: Task Cascades Overview

Input: Documents D, original operation o_{orig}, oracle and proxy models
Output: Task cascade π (optionally with statistical guarantees)

`// Prepare development set`
Sample D_{dev} from D;
`// Document Restructuring (Section 4)`
foreach *document* $x \in D_{\text{dev}}$ **do**
  Train lightweight classifier to score chunk relevance;
  Reorder x to front-load relevant content;
end
`// Initialize candidate tasks with original operation at multiple document fractions` $\mathcal{F}$
$\mathcal{T} \leftarrow \{(m, o_{\text{orig}}, f) : m \in \{\text{proxy, oracle}\}, f \in \mathcal{F}\}$;
`// Agentic Loop for Surrogate Discovery`
for n_a *iterations* **do**
  `// Assemble best cascade from current candidates (Section 3)`
  $\pi \leftarrow \text{GreedyCascadeAssembly}(\mathcal{T}, D_{\text{dev}})$;
  `// Generate new surrogate operations using agent (Section 5)`
  $O_{\text{new}} \leftarrow \text{LLMAgent}(o_{\text{orig}}, \pi)$;
  `// Add new candidates to task set`
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(m, o, f) : o \in O_{\text{new}}, m \in \{\text{proxy, oracle}\}, f \in \mathcal{F}\}$;
  if *no cost improvement* **then**
    break;
  end
end
return π;

o_{orig} appears at the beginning. This reordered layout lets us reuse computation: if a longer document fraction is needed later in the cascade, the tokens from shorter prefixes have already been processed and cached by the model, so we only pay to process the additional new content. We use the oracle model to label regions of each document by relevance to o_{orig}, then train a lightweight classifier to automate reordering for new documents not in D_{dev}.

2.2.2 Agentic Loop for Cascade Construction. To build π, we iteratively grow a set of candidate task configurations $\mathcal{T}$ and repeatedly assemble the lowest-cost cascade that meets the target accuracy α on D_{dev}. Each candidate task configuration is defined by a model, an operation, and a document fraction—that is, the fraction of the reordered document (starting from the beginning) made available to the model. We use a fixed set $\mathcal{F}$ of fractions (e.g., $\mathcal{F} = \{0.1, 0.25, 0.5, 1.0\}$). Initially, we populate the candidate set $\mathcal{T}$ with all combinations of the original operation o_{orig}, both models, and these document fractions. (Confidence thresholds for each task are determined during cascade assembly.)

Each iteration of the agentic loop proceeds as follows. First, we assemble the lowest-cost cascade, π_{best}, that achieves the accuracy target, given the current $\mathcal{T}$. Next, we identify π_{best}'s "failure cases,"

Table 1: Table of Notation

Symbol	Description
$x \in D$	A document in a collection D
D_{dev}	Development set of documents (to find a cascade)
o_{orig}	Original operation, in natural language
$c \in C$	A class from a predefined set of classes C
α	Accuracy target, where $\alpha \in (0, 1]$
m; $m_{\text{oracle}} \in \mathcal{M}$	Any model; oracle model in the set of models
$p_m(c \mid x_f, o)$	Score produced by m for class c, given doc fraction x_f and operation o
τ^c	Class-specific confidence threshold for class c
(m_i, o_i, f_i)	A task config., defined by m_i, operation o_i, doc fraction f_i
$T_i = (m_i, o_i, f_i, \tau_i)$	A task, comprising a task config. and confidence thresholds τ_i
$\pi = (T_1, T_2, \ldots, T_k)$	A task cascade, i.e., ordered sequence of tasks
$Cost(T_i, x)$	Cost to run task T_i on document x
$\lvert \cdot \rvert$	Number of tokens or size
λ_{in}, λ_{cached}	Cost per new (cached) input token
$f \in \mathcal{F}$	Set of document fractions to consider for each task
n_s; n_a	Number of new surrogate operations per iteration; number of iterations
g	Minimum fraction of D_{dev} classified by a task for inclusion in cascade

i.e., documents that reach the oracle or are not confidently classified by any task in the cascade. We then prompt an LLM agent to propose new surrogate operations, providing context about these failure cases to inform the agent's proposals. For each new surrogate, we pair it with all models and document fractions and add the resulting tasks to $\mathcal{T}$. We repeat this loop for a fixed number of rounds or until no further cost improvements are found.

Our approach has three main technical components, and we describe them in the remainder of the paper. First, we present cascade assembly (Section 3), which operates independently of how tasks in the cascade are produced. We then follow the approach to creating candidate tasks as depicted in Figure 2: document restructuring (Section 4) followed by surrogate generation (Section 5). These components are highly modular and can each be customized or replaced.

3 CASCADE CONSTRUCTION

We describe how to construct a cost-efficient cascade from a set of candidate task configurations $\mathcal{T} = \{(m_i, o_i, f_i)\}$, comprising model, operation, and document fraction tuples. Our objective is to assemble an ordered sequence of tasks, with associated confidence thresholds, that minimizes inference cost while meeting a target accuracy α, as formalized in Section 2.1. For this section, we assume $\mathcal{T}$ is fixed; Section 4 will describe how we reorder content within each document, creating fractional documents, and Section 5 will explain how we generate new surrogate operations. Throughout, we assume a fixed set of document fractions per model and operation (original or surrogate).

We first show that a substantially simplified version of cascade assembly is NP-hard (Section 3.1), motivating our greedy approach. We then detail a three-step procedure for constructing the cascade (Section 3.2): (i) filtering out candidate tasks that cannot meet the accuracy requirement; (ii) greedily assembling an ordered cascade that maintains per-task accuracy; and (iii) adjusting the cascade to meet the overall accuracy guarantee.

3.1 NP-Hardness of Optimal Task Cascade

Constructing an optimal task cascade π on D_{dev}, given task configurations $\mathcal{T} = \{(m_i, o_i, f_i)\}$ is NP-Hard. We show hardness holds even if all $m_i = m$, a single proxy model, and all fractions $f_i = 1$. Our reduction is from the NP-Hard Min-Sum-Set-Cover problem [18] (MSSC), a variant of the more standard Set-Cover problem. In MSSC, we similarly pick a subset of sets that cover a universe of items; however, the output is an ordering of these sets, where, for each item, we pay a cost based on the earliest set that covers it; specifically, if the ith set in the sequence covers item j, then the cost for covering j is i. Our goal is to minimize the total cost of covering all items.

Theorem 3.1. *Constructing an optimal task cascade is* NP-Hard.

Proof. (Sketch) Given an instance of MSSC, (U, S), where U denotes the items, and S denotes the sets, we generate an instance of cascade assembly as follows. For each item $u \in U$, we create a document d_u. For each set $S_i \in \mathcal{S}$, we create a candidate task $T_i = (m, o_i, 1)$, where m is a single proxy model, and o_i is an operation that predicts True with confidence 1 on d_u iff $u \in S_i$ and returns a random answer with confidence 0 otherwise. Therefore, every task in our cascade will have thresholds set to 1.

We design the cost model so that accessing cached document tokens costs 0; each document is cached after the first task that processes it. Thus, the total cost for processing document tokens is constant across all cascades and can be ignored. We set each operation o_i to have cost 1, so running any task T_i on any d_u incurs cost 1. We set the accuracy target $\alpha = 1$ and assign the oracle infinite cost, so every document must exit the cascade via some task. Under this cost model, processing any d_u through a cascade $(T_{i_1}, T_{i_2}, \ldots)$ incurs a cost equal to the index of the first T_{i_j} with confidence 1 on d_u. Hence the cascade cost equals the MSSC objective. □

Feige et al. [18] describe a greedy algorithm for MSSC, which, at each stage, picks the set that covers the most (uncovered) items, and has an approximation factor of 4. Feige et al. demonstrate that further improvements to the approximation factor are difficult. We present an analogous greedy algorithm, adapted to handle varying models, operations, costs, document fractions, and accuracies.

3.2 Assembling a Task Cascade

At a high level, assembling a task cascade from a fixed candidate set of task configurations $\mathcal{T}$ consists of the following steps:

(1) **Threshold selection and filtering.** For each candidate task individually, we find the smallest confidence thresholds such that at least g% of D_{dev} are classified by the task, with accuracy at least α. We remove any task for which both criteria are not met for any class.
(2) **Cascade assembly.** We build a cascade by greedily adding the task that most reduces total inference cost at each step, such that each task achieves the target accuracy on the subset of documents it classifies.
(3) **Threshold adjustment.** We adjust the thresholds in the assembled cascade to ensure that the overall sequence meets the desired accuracy.

We discuss each of these steps in turn.

3.2.1 Threshold Selection and Filtering. Our candidate set $\mathcal{T}$ may contain hundreds of task configurations, many of which have no chance of meeting the target accuracy, e.g., tasks based on poor surrogate operations. We eliminate these by discarding any task configuration that individually cannot confidently classify enough documents at the required accuracy. For the remaining, we determine confidence thresholds up front; these thresholds are used in subsequent cascade assembly.

We apply the procedure in Algorithm 2: for each candidate task, we examine its predictions on D_{dev} and, for each class, find the lowest confidence threshold such that predictions above this threshold achieve the target accuracy α. If no such threshold exists for a class (i.e., the required accuracy cannot be achieved for any threshold), we set it to be ∞, ensuring any documents that are predicted to be this class by the task do not exit the cascade. If, across all classes, these thresholds allow the task to classify at least a minimum fraction g of D_{dev}, we retain the task and record its thresholds; otherwise, we discard it. In our implementation, we set g to be 10%; g could also be set using statistical bounds (e.g., from Hoeffding's inequality), but for high accuracy targets, we would require many examples per task (e.g., > 250 for $\alpha = 0.95$ and $\delta = 0.25$). We still ensure statistical guarantees on our eventual cascade, as we will discuss in Section 3.2.3.

Algorithm 2: Find Thresholds for a Given Task Config.

Input: Candidate task config. $T = (m, o, f)$, development set D_{dev}, accuracy target α
Output: Per-class thresholds $\{\tau^c\}$, or discard T

Initialize $\tau^c \leftarrow \infty$ for all c; `total` $\leftarrow 0$;
foreach *class c* **do**
  $P_c \leftarrow$ confidence scores assigned to class c by T on all $x \in D_{\text{dev}}$ where c is the model's predicted class for x (i.e., $\arg\max_{c'} p_m(c' \mid x_f, o) = c$); sorted by confidence;
  foreach *unique t in P_c (asc)* **do**
    $S \leftarrow \{x \in D_{\text{dev}} : \arg\max_{c'} p_m(c' \mid x_f, o) = c,\ p_m(c \mid x_f, o) \geq t\}$;
    if $|S| > 0$ ***and*** *accuracy of assigning S to c* $\geq \alpha$ **then**
      $\tau^c \leftarrow t$; `total` $+= |S|$;
      break
    end
  end
end
return $\{\tau^c\}$ **if** `total` $\geq g \cdot |D_{\text{dev}}|$; **else** discard T;

3.2.2 Cascade Assembly. After filtering, we build the cascade greedily (detailed pseudocode is in Algorithm 4). We start with the empty cascade π_0 that invokes the oracle on all documents. At each step, we consider inserting each unused candidate task at the end of π, creating a new candidate cascade π'. For each such candidate cascade π', we execute it on D_{dev} to compute its cost, and to measure the accuracy of every task in π' on the subset of documents it classifies (i.e., those reaching that task, with the output having confidence above its threshold). We only consider π' where every task, including the newly added one, achieves the target accuracy α on its assigned subset. Among those that meet the per-task accuracy requirement, we select the one with the lowest total cost. If no such candidate cascade reduces the cost relative to the current cascade, we return the current cascade.

By requiring every task in the cascade to independently satisfy the target accuracy on its assigned documents, we may construct a more conservative cascade than strictly necessary. However, since D_{dev} may be small, a cascade that merely fits the overall accuracy constraint on the development set may not generalize at inference time. Enforcing per-task accuracy thus increases the likelihood that the cascade meets the desired target. Note that for simplicity, this greedy assembly algorithm operates on tasks with fixed, pre-determined confidence thresholds (set in Section 3.2.1). The subsequent threshold adjustment step (Section 3.2.3) is purely a post-processing step to provide statistical accuracy guarantees, and does not further optimize cascade cost.

Algorithm 3: Threshold Adjustment

Input: Cascade thresholds $\mathcal{T}_0$ and validation set D_V
Output: Cascade thresholds guaranteed to meet the target accuracy

for *iteration i until maximum iteration* max_iter **do**
  $\mathcal{T}_i \leftarrow \{\tau_j + \epsilon_j^i; \forall \tau_j \in \mathcal{T}_0\}$
  if *not* $\mathcal{E}(\mathcal{T}_i, D_V)$ **then**
    if $i > 1$ **then**
      return $\mathcal{T}_{i-1}$;
    end
    else
      return *Not Found*;
    end
  end
end
return $\mathcal{T}_{max\_iter}$;

3.2.3 Meeting Accuracy Targets with Guarantees. The cascade π assembled so far is only empirically accurate on the development set and may not achieve the desired accuracy α on new data. To obtain statistical guarantees, we randomly partition the development set D_{dev} into two i.i.d. splits of equal size: a "training" split D_T, used to construct the cascade π, and a "validation" split D_V, used to adjust thresholds and yield a final cascade π^* such that $\Pr[\text{Acc}(\pi^*) < \alpha] \leq \delta$. If D_T or D_V are too small or highly variable, the procedure may simply fail to find any cascade meeting the target accuracy under the specified bound; accordingly, our experiments use at least 75 documents in each split. The new task cascade $\pi^* = (T_1^*, ..., T_k^*)$ is composed of tasks $T_i^* = (m_i, o_i, f_i, \tau_i^*)$ where for all tasks $T_i = (m_i, o_i, f_i, \tau_i)$, m_i, o_i and f_i are kept as is, but the thresholds τ_i are adjusted to τ_i^*. For convenience, we will refer to $\mathbf{t} = \{\tau; \tau \in \tau_i^C,\ \forall C\ \forall i\}$ as the set of all thresholds in all tasks in π.

At a high level, determining thresholds that guarantee the target accuracy proceeds as follows. We start by incrementing each threshold in $\mathbf{t}$ by a large non-negative offset $\epsilon_j^{(1)}$, producing an initial, highly conservative set of thresholds. We then iterate through (1) *adjusting* this offset, and (2) *estimating* whether the resulting thresholds are sufficient. In the adjustment phase, at each iteration i, we reduce the offsets to create a new candidate threshold set $\mathbf{t}^{(i)} = \tau_j + \epsilon_j^{(i)} : \tau_j \in \mathbf{t}$, where the adjustment strategy $\epsilon_j^{(i)}$ is chosen to decrease with each step; i.e., $\epsilon_j^{(i+k)} \leq \epsilon_j^{(i)}$ for $k \geq 0$. We will discuss our exact adjustment strategy in Section 3.2.4. In the estimation phase, we run the cascade with $\mathbf{t}^{(i)}$ on the validation split D_V and apply an estimation function $\mathcal{E}(\mathbf{t}^{(i)}, D_V)$. This function returns `True` if the cascade achieves accuracy at least α on D_V with failure probability at most δ, and `False` otherwise. This procedure is presented in Algorithm 3, where the two steps of *adjust* and *estimate* are repeated until a possible maximum number of iterations, and we stop at the first instance where the estimation function determines that the thresholds do not meet the target. If no candidate set passes, we revert to the oracle-only cascade (which never happens in our experiments).

Theorem 3.2. *For any adjustment strategy ϵ and appropriate estimator $\mathcal{E}$, Algorithm 3 returns a cascade π^* with* $\Pr(\mathrm{Acc}(\pi^*) < \alpha) \leq \delta$.

Theorem 3.2 shows that Algorithm 3 yields the desired accuracy guarantee for any valid estimator and monotonic adjustment schedule. A valid estimator $\mathcal{E}$ must provide a statistical test that, given a candidate set of thresholds and a validation set, returns `True` only if it can certify—based on the observed sample—that the cascade achieves accuracy at least α with failure probability at most δ. This is a classic problem of mean estimation from i.i.d. Bernoulli samples, and any sound concentration bound can be used; e.g., Hoeffding's inequality. We adopt the estimator from Waudby-Smith and Ramdas [66], which uses both the sample mean and variance to produce tighter bounds—particularly when the variance is small—compared to Hoeffding's inequality, which relies only on the mean. The proof of Theorem 3.2 and the full specification of $\mathcal{E}$ are given in Appendix A. We also account for multiple applications of $\mathcal{E}$ within the adjustment loop, ensuring the total probability of failure remains bounded by δ.

3.2.4 Threshold Adjustment Strategy. We now describe our strategy for adjusting thresholds to ensure the final cascade meets the desired accuracy guarantee. A naive approach is to decrement each threshold by a fixed amount at every iteration (e.g., -0.01 per step). However, LLM-generated confidence scores are often poorly calibrated and heavily concentrated near 1, making uniform decrements ineffective [14, 29, 32]. Instead, for each class c of each task $T = (m, o, f, \tau)$, we collect all confidence scores greater than the initial threshold τ^c assigned by the model to predictions of class c on the training split D_T. We sort these confidence scores in ascending order: $\tau^c < p_1 < p_2 < \cdots < p_k$, where p_1 is the lowest confidence just above τ^c and p_k is the highest.

We introduce a *shift index* s that starts at $s_{\max}$ and decrements by 1 at each iteration. At each iteration, for every class, we set its threshold to p_s, corresponding to a more conservative cutoff when s is large, and a less conservative one as s decreases. Specifically, we start with $p_{s_{\max}}$ (the most conservative threshold), and at each subsequent iteration decrement s by 1, thereby shifting the threshold closer to the original τ^c. Once $s = 0$, we use the original threshold τ^c. After updating all thresholds in this way for an iteration, we evaluate the cascade on D_V and apply the estimator $\mathcal{E}$ to check the accuracy guarantee. If the guarantee is met, we continue; if not, we return the thresholds from the previous shift. Our complete algorithm is given in Appendix A.3. In our implementation, we set $s_{\max} = 5$, but values between 3 and 10 are generally effective.

4 DOCUMENT RESTRUCTURING

Tasks in a cascade can process different fractions of each document, allowing models to focus on the most relevant content for the user-defined operation. To reduce cost under our prefix-based cost model (Section 2.1), we reorder each document so that the most relevant text appears first. This restructuring step is independent of the choice of surrogate operations or cascade configuration; any method for ranking document segments by relevance can be used.

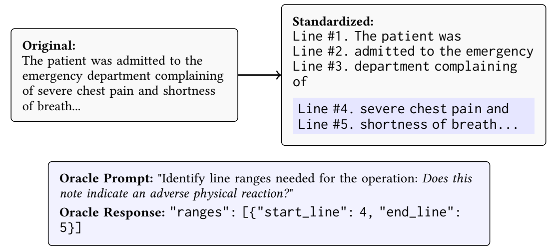

Figure 3: Standardizing documents. This format allows the oracle to specify relevant content as line ranges.

While document restructuring is optional, skipping it forgoes the prefix-caching discounts that substantially reduce cost.

To restructure documents effectively, we first choose how to segment them. Some user operations require only short spans of text (e.g., a sentence), while others need broader context. We represent each document as a sequence of lines and select a segmentation *granularity*—the number of consecutive lines to treat as a unit. Each document is then divided into contiguous *chunks* of this granularity. Then, we train a lightweight classifier to score and rank chunks in a document by their relevance to the user-defined operation. The following sections describe how we determine the chunk granularity and train the relevance classifier.

Determining Chunk Granularity. To select the chunk granularity for the user-specified operation o_{orig}, we use the oracle LLM and the development set D_{dev}:

(1) We split each document into lines of 80 characters, each prefixed with a line number (see Figure 3).
(2) For each document, we ask the oracle LLM to return the minimal set of non-overlapping line ranges needed to answer o_{orig}. The oracle returns these line ranges in a structured format (e.g., `"start_line": 5, "end_line": 12`).
(3) We merge overlapping or adjacent line ranges and create a reduced document containing only the union of these lines. We run the oracle LLM on this reduced version and check whether its answer matches the answer on the full document.
(4) If, across D_{dev}, this reduced document yields the correct answer for at least an α fraction of documents, we move to (5). Otherwise, we expand each range by one line before the start and one line after the end (while staying within document boundaries), merge again, and repeat the check. This process continues until the target accuracy is achieved or a maximum of $e = 3$ expansions.
(5) We set the chunk granularity to the average length, in lines, of the final merged ranges across all documents in D_{dev}.

Consider the following example for a document x to illustrate the expansion and merging procedure. Suppose the oracle initially selects the ranges `[23,25]` and `[28,30]`, with average length $(3 + 3)/2 = 3$. If the answers produced by running o_{orig} on the selected lines do not match the answers on the full document for at least an α fraction of documents in D_{dev}, we expand each range for x by one line at both ends, resulting in `[22,26]` and `[27,31]`, with average length $(5 + 5)/2 = 5$. If another expansion is needed, we obtain `[21,27]` and `[26,32]`, which now overlap and are merged into a single range, `[21,32]`, with average length 12.

Scoring Chunks by Relevance. Once we determine the segmentation granularity s, we divide each document into contiguous sets of s lines—referred to as *chunks*. Our goal is to train a lightweight *relevance classifier* that predicts whether a chunk contains information required to perform o_{orig}.

We first partition D_{dev} into "training" (D_{train}) and "held-out test" (D_{test}) splits for constructing and evaluating the classifier. From the previous step, we already have starting lines of relevant chunks for all documents. We construct an oracle-labeled dataset for training the relevance classifier as follows:

(1) For each starting line identified by the oracle, we extract the s-line chunk beginning at that line and label it as relevant ($r = 1$). To construct irrelevant examples, we slide a non-overlapping window of s lines from the start of the document, and include a chunk as irrelevant ($r = 0$) only if it does not overlap with any relevant chunk. Chunks that overlap with any relevant chunk are excluded from the irrelevant set.
(2) For each document in D_{train} and D_{test}, we collect all labeled relevant and irrelevant chunks as described above. The training and test datasets are formed by taking the union of all labeled chunks from each document in D_{train} and D_{test}, respectively.

Each chunk c in D_{train} and D_{test} is embedded using OpenAI's `text-embedding-3-small`, producing pairs $(\text{embedding}(c), r)$. Because relevant chunks are rare, we upsample the relevant class during training. We fit a logistic regression model with weights initialized to the embedding of o_{orig} to predict the relevance for each chunk. The model is trained to minimize binary cross-entropy loss on the chunks from D_{train}, using the Adam optimizer and early stopping based on F1 score on the chunks from D_{test}.

At inference time, a new document is partitioned into consecutive, non-overlapping chunks of s lines; each chunk is embedded and assigned a probability of relevance by the trained model. Chunks are then sorted by predicted probability (from highest to lowest) and concatenated to yield the reordered document.

5 SURROGATE OPERATION GENERATION

So far, we have described how to assemble a cost-optimal cascade from a fixed set of candidate task configurations. We now address the question of *how to create this candidate set*. We assume that all documents have already been reordered so that content most relevant to the original operation appears at the beginning (see Section 4 for details), and we only discuss how to generate surrogate operations. At a high level, our approach proceeds as follows. We initialize $\mathcal{T}$ to include all combinations of o_{orig}, each model $m \in \mathcal{M}$, and each fraction $f \in \mathcal{F}$. We then enter a loop that, at each round, (1) assembles the best cascade from the current $\mathcal{T}$, (2) analyzes the failure cases of this cascade, and (3) expands $\mathcal{T}$ using an LLM agent. We next explain what constitutes a valid surrogate operation and how surrogate operations are elicited from the agent, and finally, how the agentic loop is implemented.

Valid Surrogate Operations. In order to generate surrogate operations, we require some notion of validity. A valid surrogate operation is any operation whose outputs form a subset of the original operation's classes. For classification tasks, a valid surrogate operation is simply any operation whose output space is a subset of the original classes (optionally including a special "none of the above" or -1 label). Suppose the original task is to classify biomedical abstracts into four classes: 0 (Research Article), 1 (Review Article), 2 (Clinical Trial), and 3 (Case Report). A surrogate operation could be: *"If the abstract contains phrases like "randomized trial" or "double-blind," output 2 (Clinical Trial); otherwise, output* -1*."* Such heuristics enable proxy models to confidently resolve certain examples with simple patterns, even if they cannot perform full multi-class classification.

Eliciting Surrogate Operations from the Agent. Using the aforementioned notion of surrogate operations, whenever we query the LLM agent for surrogates, we include the following context:

(1) The user-defined operation, o_{orig}.
(2) "Failure cases," or examples of documents that are *not* classified by any existing proxy task in the current best cascade (i.e., documents that reach the oracle). To ensure we can fit multiple examples of documents within the agent's prompt window, we use the oracle model to identify and concatenate only the most relevant snippets from each document—those that support or explain the oracle's prediction—rather than the full document.
(3) For each task in the current cascade: summary statistics including whether it was selected into the cascade, its document coverage (i.e., number of documents it resolves), and up to 10 "hard examples," or borderline cases where the task predicted incorrectly with high confidence, typically those with confidence just above the acceptance threshold.
(4) Explicit instructions to propose new surrogate operations that (i) are simpler than the original, (ii) output any subset of the original classes, or (iii) target weaknesses in the current cascade. The prompt includes concrete examples and specifies an output format (`SURROGATE PROMPT: ...`, `RATIONALE: ...`), so we can parse the surrogate operations. See Appendix B for details.

Agentic Loop. Now that we have a mechanism for eliciting surrogate operations from the agent, we describe the full agentic loop. At each round: (1) We assemble the best cascade using the current candidate set, via the procedure in Section 3.2; (2) We update the agent's prompt with the new round's failure cases and summary statistics; and (3) The agent proposes n_s new surrogate operations. Each surrogate is paired with every model and document fraction, resulting in $|\mathcal{M}| \times |\mathcal{F}| \times n_s$ new candidate tasks, which are added to $\mathcal{T}$. We repeat this process until n_a rounds have completed or no further cost improvement is observed. Parameter choices (e.g., n_s, n_a, document fractions, agent model) are flexible (see Section 2); we $n_s = 3$, $n_a = 3$, and OpenAI's o1-mini as the agent. See Appendix B for the full algorithm.

6 COST MODEL FOR OPTIMIZATION

In this section, we derive the optimization cost for constructing task cascades and show how hyperparameter choices can control the optimization budget. The total optimization cost comprises three components: document restructuring, surrogate operation generation and evaluation, and agent loop costs. For simplicity, we omit candidate task output token costs (which are constant and negligible—e.g., 1 token for binary classification, at most a few tokens for multi-class tasks). Note that in practice, the actual optimization cost may be lower than our derivation suggests: LLM API providers employ prompt caching, and evaluating multiple candidate tasks on the same documents can lead to cache hits that reduce costs. Our optimization implementation does not explicitly optimize for cache reuse, so the formulas below represent a conservative estimate on the true optimization cost.

Initial labeling and document restructuring. We first run the oracle model once on the full development set to obtain "ground truth" labels for computing surrogate task accuracy: $C_{\text{labels}} = N(L+P)\lambda_o$, where N is the number of development documents, L is the average number of tokens per document, P is the number of prompt tokens per call, and λ_o is the oracle model price per token.

Next, we perform document restructuring to prepare documents for fractional processing. We run the oracle model once more to identify relevant line ranges or chunks of the document. We also generate embeddings for all chunks:

$$C_{\text{doc}} = C_{\text{labels}} + N(L+P)\lambda_o + NL\lambda_{\text{emb}} = N(L+P)(2\lambda_o) + NL\lambda_{\text{emb}}$$

where λ_{emb} is the embedding price per token.

Surrogate operation generation and evaluation.. Over n_a iterations, the agent proposes n_s new surrogate operations per iteration, yielding $n_s n_a$ total surrogates. Each surrogate is paired with each document fraction $f \in \mathcal{F}$ and evaluated on the development set using both the proxy and oracle models. Since we evaluate at $|\mathcal{F}|$ different fractions, the total fractional content processed per document is $S_f = \sum_{i=1}^{|\mathcal{F}|} f_i$ (e.g., for $\mathcal{F} = \{0.15, 0.25, 0.5, 1.0\}$, we have $S_f = 1.90$). With λ_p the proxy model price per token, the cost for all surrogate evaluations is:

$$C_{\text{eval}} = Nn_s n_a \left[LS_f(\lambda_o + \lambda_p) + P|\mathcal{F}|(\lambda_o + \lambda_p)\right]$$

Agent loop. The LLM agent (o1-mini) generates new surrogates over n_a iterations, consuming T_{agent} input tokens and producing O_{agent} output tokens per round, with prices $\lambda_{o1,\text{in}}$ and $\lambda_{o1,\text{out}}$. Notably, this cost is independent of the development set size N and thus constant: $C_{\text{agent}} = n_a(T_{\text{agent}}\lambda_{o1,\text{in}} + O_{\text{agent}}\lambda_{o1,\text{out}})$.

Total optimization cost. The total optimization cost is $C_{\text{opt}} = C_{\text{doc}} + C_{\text{eval}} + C_{\text{agent}}$. The dominant component is C_{eval}, the cost of evaluating candidate surrogate tasks on the development set. This cost can be reduced by: (1) decreasing n_s or n_a to generate fewer surrogates, (2) using fewer document fractions (smaller $|\mathcal{F}|$), (3) excluding the oracle model when generating candidate tasks (replacing $\lambda_o + \lambda_p$ with λ_p in C_{eval}), or (4) reducing the development set size N, though (4) makes statistical guarantees harder to achieve. In our experiments, we also evaluate a cheaper variant using strategy (3).

7 EVALUATION

We design our experiments to answer the following questions:

Q1 Do task cascades reduce inference cost compared to model cascades, and which components of our approach contribute most to these savings?
Q2 How do cost and accuracy change as target accuracies vary?
Q3 How consistent are task cascade costs and accuracy across runs?
Q4 What is the cost of building task cascades?

Across all workloads at a 90% target accuracy, ***our approach reduces inference cost by 48.5% on average compared to model cascade baselines (with comparable guarantees), and by 86.2% compared to using the oracle model alone***, highlighting the effectiveness of task cascades (**Q1**). Every component of our approach (i.e., surrogate operation discovery, document pruning, and cascade ordering) contributes to cost reduction. For **Q2**, task cascades consistently improve the cost–accuracy tradeoff as the target accuracy varies. The largest gains occur at lower target accuracies, likely because it is easier to identify surrogate operations that satisfy relaxed accuracy targets. On more challenging workloads, the performance gap between model cascades and task cascades narrows at higher accuracy targets. For **Q3**, we find that task cascades and model cascades both exhibit high variance, but task cascades consistently achieve lower mean and median costs. Finally, for **Q4**, we show that optimization costs are negligible at scale, and that even considering optimization cost, task cascades are cheaper than model cascades.

We first describe our experimental setup (Section 7.1). Next, we present results for each research question: cost-effectiveness across variations (Section 7.2.1), performance under varying accuracy targets (Section 7.2.2), consistency across repeated runs (Section 7.2.3), and optimization costs and break-even analysis (Section 7.2.4). Finally, we provide insights from analyzing task cascades, including practical deployment considerations and parameter selection guidance (Section 7.3). We also detail examples of surrogate tasks and the number of tasks in each cascade in Appendix C.

Table 2: Workloads with average word counts and corpus sizes. "# Docs" reports the approximate number of documents in the dataset at release time; actual corpus sizes may have changed in later versions.

Dataset	Description of o_{orig}	Avg. words	# Docs
AGNEWS	Classify news article summaries into one of four topics: *World*, *Sports*, *Business*, or *Science/Tech.*	~37	~128k
COURT	Determine if a U.S. Supreme Court opinion reverses the lower-court ruling.	~3.7k	~36k
ENRON	Identify emails sent by C-suite or VP-level executives in the Enron corpus.	~1.5k	~500k
FEVER	Decide whether a natural-language claim is supported by the provided evidence snippets.	~5.1k	185k
GAMES	Determine whether a review praises a different game more than the one being reviewed.	~1.1k	~6.4M
LEGAL	Detect covenants not to sue or IP no-challenge clauses in license agreements.	~8.0k	510
PUBMED	Classify biomedical articles into one of six study types: *RCT*, *Observational*, *Meta-analysis*, *Bench/Lab*, *Computational*, or *Review.*	~3.1k	~133k
WIKI_TALK	Predict whether a Wikipedia Talk-page discussion culminates in an edit revert.	~0.9k	~125k

7.1 Setup

7.1.1 Datasets and Workloads. We evaluate our approach on eight document classification workloads chosen to span a range of domains, document lengths, and task complexities (Table 2). Five are drawn from prior research on LLM-powered data processing systems. To broaden coverage, we introduce three additional tasks from Kaggle that require multi-step or domain-specific reasoning.

From Prior Work. Workloads (i)-(ii) are derived from prior work on model cascades [9, 49], while (iii)-(v) are from work on LLM-powered data processing [41, 53]: (i) AGNEWS [13], from [9], is a four-way classification task with extremely short documents. We include AGNEWS as an example of a simple workload where the proxy already meets the accuracy target, to test whether task cascades can provide additional benefits. (ii) FEVER [59], from [49], is a claim verification task: given a natural-language claim and a set of evidence snippets, the LLM must decide whether the claim is supported. Since each document involves different claims and evidence, reusable filters and surrogate tasks are difficult to learn, making FEVER particularly challenging for our approach. (iii) ENRON [38], from [41], consists of emails from the Enron corpus. The original task in [41] involved fraud detection, but fraud cases are extremely rare in the dataset, so a trivial "always predict no fraud" strategy achieves very high accuracy. To create a more balanced classification problem suitable for evaluating task cascades, we adapt the original fraud detection task to identifying emails sent by C-suite or VP-level executives, which provides a more even class distribution. (iv) LEGAL [26], from [53], tests detection of specific legal clauses in contracts; we convert the original span extraction task into binary classification by focusing on the presence or absence of a single clause type. (v) GAMES [56], from [53], is adapted to a more challenging classification task: determining whether a video game review praises another game more than the one being reviewed.

New Workloads. We introduce three additional workloads from Kaggle and recent NLP datasets. COURT [22] involves classifying U.S. Supreme Court opinions and requires multi-step reasoning over lengthy, complex texts. WIKI_TALK [6, 12] challenges models to predict whether a Wikipedia Talk-page discussion culminates in an edit revert, capturing dynamics of online discourse. Finally, PUBMED [11] is a multi-class classification task on long biomedical articles, testing our approach under both large document size and an expanded label space.

For each workload, we sample 1,000 documents (200 for development, 800 for test), except LEGAL, which contains only 509 documents (150 development, 359 test). We sampled datasets to stay within a $5,000 API budget; running all variants described below across full datasets with multiple accuracy targets and trials would have cost hundreds of thousands of dollars. All workloads represent binary classification tasks except AGNEWS and PUBMED, which are multi-class. Prompts for all workloads are in Appendix F.

7.1.2 Baselines. For all experiments in **Q1–Q4**, we compare task cascades against three reference approaches. `Oracle Only` runs the oracle model (GPT-4o) on every document, giving an ideal but costly upper bound. Our second baseline, `2-Model Cascade`, uses the model cascade approach from [49], pairing GPT-4o-mini as proxy with GPT-4o as oracle. This approach is state-of-the art as it leverages token-level log probabilities from LLM APIs, unlike prior model cascade papers that employ more heavy-weight and less accurate approaches [9, 70]. For each class, we set the proxy threshold to the smallest value on the development set such that the combined accuracy of proxy predictions above the threshold *and* oracle predictions below it meets the target accuracy, aggressively minimizing cost. Third, `2-Model Cascade (+ Guarantees)` augments `2-Model Cascade` with our statistical guarantee procedure (Algorithm 3). Although alternatives such as SUPG [35] could enforce guarantees, we apply the same procedure we do to ensure a controlled comparison.

For **Q2**, which evaluates cost–accuracy tradeoffs across varying accuracy targets, we additionally include `LOTUS (+ Guarantees)` [49]. LOTUS embeds model cascades inside filter operators but provides guarantees on *precision* and *recall*, rather than overall accuracy. Because LOTUS requires separate precision and recall targets instead of a single accuracy target, we cannot directly compare it to our approach at fixed accuracy levels (as in **Q1** and **Q3**). For **Q2**, we implement the LOTUS baseline using their `sem_filter` operator and evaluate it across all combinations of precision and recall targets in $\{0.60, 0.80, 0.90, 0.95\} \times \{0.60, 0.80, 0.90, 0.95\}$ on each binary classification workload.

7.1.3 Variants Compared. Like our baselines, all methods use a single proxy model, along with the oracle model. We evaluate `Task Cascades`, our main approach, as well as `Task Cascades (+ Guarantees)`. We also evaluate `Task Cascades (Lite)`, a lightweight variant that minimizes optimization cost by using the same parameters as `Task Cascades` but evaluating all candidate surrogate tasks only with the proxy model, never the oracle, substantially reducing C_{eval} (as described in Section 6). Note that we distinguish `Task Cascades`, the approach, from task cascades, the outcome of such an approach; throughout this section, `Task Cascades` refers to our approach and is treated as a singular entity (e.g., "`Task Cascades` achieves lower cost"), while a "task cascade" refers to an individual cascade configuration discovered by the approach. We also include different variants of task cascades to isolate the effect of each component of our approach:

(1) **Surrogate Operation Discovery Variants.** `No Surrogates` disables surrogate discovery, so cascades include only the original user-specified operation at different document fractions. `Single-Iteration` generates all surrogate operations in a single batch, without iterative refinement.
(2) **Document Pruning Variants.** `No Filtering` disables learned document pruning, so all tasks must process full documents. `Naive RAG Filter` replaces learned pruning with a simple retrieval filter that ranks chunks by cosine similarity to the embedding of the user-defined operation.
(3) **Alternative Cascade Designs.** Rather than using the default greedy strategy, `Selectivity Ordering` constructs the cascade by prioritizing operations with the highest (selectivity -1)/cost ratio, as in prior work on predicate ordering [25], where selectivity is the number of documents not classified by the task and thus passed down the cascade. `Restructure (Top-25%)` applies our learned document restructuring to reorder each document, then allows the proxy to process only the top 25% of the restructured content, while the oracle operates on the full document as usual. This approach isolates the effect of learned restructuring and pruning with a 2-task cascade, without introducing surrogate discovery or having more than 2 tasks. Finally, `RAG + NoSur` applies only naive retrieval-based pruning with no surrogate discovery. This variant allows us to quantify the benefit of simple retrieval and isolate the added value of task cascades.

7.1.4 Metrics and Implementation Details. We report average inference cost in USD ($) and the fraction of workloads where each method meets the accuracy target α. Most experiments use a fixed $\alpha = 0.9$; in one experiment, we vary α from 0.75 to 0.95. For methods with guarantees, we set failure probability $\delta = 0.25$.

We use OpenAI models: GPT-4o as the oracle model, GPT-4o-mini as the proxy, o1-mini for surrogate generation, and `text-embedding-3-small` for embeddings. We use PyTorch for the relevance classifier in Section 4. All LLM calls use temperature 0. We consider four document fractions $\mathcal{F} = \{0.1, 0.25, 0.5, 1.0\}$ and run surrogate discovery for three iterations ($n_a = 3$) with three surrogate operations per iteration ($n_s = 5$). These parameter values were held constant across all workloads. While more iterations or surrogates may yield cheaper cascades at higher offline cost, we find these defaults work well across diverse tasks, and run more experiments in Appendix D to demonstrate that task cascades are robust to parameter choices.

Inference costs are computed using OpenAI API pricing: $2.50/1M input tokens and $10.00/1M output tokens for GPT-4o, $0.15/1M input tokens and $0.60/1M output tokens for GPT-4o-mini, with a 50% discount for prefix-cached completions. Embedding costs using ($0.02/1M tokens) are also included in all reported inference costs; these represent about 0.13× the cost of GPT-4o-mini inference and are negligible compared to overall LLM inference costs. All methods were implemented in Python and run on a 2024 MacBook Air (M4 chip) with OpenAI API calls, costing over $3,000 USD.

7.2 Results

We now present experimental results addressing the questions outlined above. Table 3 presents the main cost and accuracy results. We reflect on optimization costs in Section 7.3 and leave a detailed discussion of these costs and latencies to Appendix E.

7.2.1 Q1: Cost Savings and Component Importances. `Task Cascades` reduces inference cost by 41% compared to `2-Model Cascade`, and 48.5% when considering accuracy guarantees. `Task Cascades (Lite)`, our lightweight variant with reduced optimization cost, achieves nearly identical savings (0.62× vs. 0.59× on average) while requiring 75% less optimization investment. Savings are most substantial on workloads with significant irrelevant content (e.g., ENRON and LEGAL); inference costs drop by up to 6× relative to `2-Model Cascade`.

Meeting the Accuracy Target. `Task Cascades` and `Task Cascades (+ Guarantees)` each miss the 90% accuracy target on one workload by < 1%, while `2-Model Cascade` fails on 3 of 8 workloads. When accuracy guarantees are required (+ Guarantees variants), both approaches construct cascades using only half the available data, reserving the remainder for threshold adjustment (Algorithm 3), which increases costs by 1.43× on average for `Task Cascades`.

Component Contributions. All three core components—greedy task ordering, surrogate operation discovery, document pruning—are essential for robust performance. `Selectivity Ordering` performs worst (7.5× worse than `Task Cascades`). `Single-Iteration` (1.13× worse than `Task Cascades`) demonstrates that iterative refinement contributes to cost reduction beyond single-round surrogate generation. Surprisingly, both `No Surrogates` (1.21×) and `No Filtering` (1.55×) are *more* expensive than the 2-Model baseline. Without surrogates, the proxy is limited to the user-defined operation and provides savings on only half the workloads; on remaining tasks, the entire document must be processed, negating cost benefits. Without document pruning, each surrogate must operate on the entire document, making each surrogate operation as expensive as the first stage of `2-Model Cascade`. Similarly, `Restructure (Top-25%)` fails to improve cost on 6 of 8 workloads: when the proxy receives too little context to perform the original operation, it cannot confidently resolve documents, escalating more to the expensive oracle. Similarly, `Restructure (Top-25%)` fails to improve cost on 6 of 8 workloads: when the proxy receives too little context to perform the original operation, it cannot confidently resolve documents, escalating more to the expensive oracle. Across workloads, simpler variants can fail in significant ways: `No Filtering` is up to 8× worse than `Task Cascades`, `No Surrogates` up to 4× worse, and the maximum degradation across all variants averages 12× worse than `Task Cascades`. By combining multiple techniques, `Task Cascades` achieves robust cost savings across diverse workloads.

Comparison with Retrieval-Based Pruning. Our learned document pruning outperforms simple retrieval alternatives. `Naive RAG Filter` achieves the second-lowest cost on average, but `Task Cascades` reduces cost by an additional 11%. More notably, `RAG + NoSur` (retrieval-based pruning without surrogate discovery) costs 1.16× more than `2-Model Cascade` on average: naive document reordering can make restructured documents harder for the proxy to process, escalating more documents to the expensive oracle. Adding surrogate discovery (`Naive RAG Filter`) addresses this degradation, making it cheaper than `2-Model Cascade` on all workloads.

Accuracy-Cost Tradeoffs at Fixed Target. At the fixed 90% accuracy target, task cascades explore a broader configuration space, enabling more precise cost-accuracy tradeoffs. In some cases, task cascades identify plans that are both cheaper *and* more accurate than baselines: on ENRON, `Task Cascades` achieves 95.5% accuracy at 0.11× cost vs. `2-Model Cascade`'s 89.1% accuracy at 1.0× cost. In other cases, task cascades find cheaper plans closer to the target: on FEVER, `Task Cascades` achieves 90.6% accuracy at 0.54× cost vs. `2-Model Cascade`'s 94.4% accuracy at 1.0× cost. Overall, `Task Cascades` can search over more configurations and, many times, land closer to the target accuracy.

7.2.2 Q2: Accuracy–Cost Tradeoffs Across Targets. To evaluate performance across different accuracy requirements, we vary the target accuracy from 75% to 95% in 5% increments, using the same workload groups as in Section 7.2.3. Figure 5 shows the resulting accuracy–cost tradeoff. We include `LOTUS (+ Guarantees)` on binary classification workloads (all except AGNEWS), using the setup described in Section 7.1. Overall, `Task Cascades` reduces inference costs and extends the Pareto frontier across both easy and hard tasks, compared to all baselines. `LOTUS (+ Guarantees)` performs comparably to `2-Model Cascade (+ Guarantees)` on all workloads. On easier Group A tasks (ENRON, LEGAL), `Task Cascades` consistently dominates the Pareto frontier, matching or exceeding model cascade accuracy at much lower cost for any target. For more challenging Group B workloads (GAMES, COURT), `Task Cascades` provides the greatest cost savings at lower target accuracies (75%—85%), because the agent can choose from a larger set of surrogate operations that meet these relaxed accuracy targets. As the target accuracy increases, fewer surrogates will satisfy the stricter requirements, so cost advantages diminish—but `Task Cascades` still often matches or outperforms the baseline, perhaps due to document pruning. On Group C (AGNEWS), where the proxy already achieves high accuracy, there is little room for improvement: `Task Cascades` performs similarly to the baseline at high targets but can identify lower-cost solutions at lower accuracy targets.

7.2.3 Q3: Consistency and Variance Across Repeated Trials. We next evaluate the consistency of task cascade outcomes across repeated

Table 3: Inference results at a 90% accuracy target on a sample of 1k documents per workload (for datasets with more than 1k documents). Each cell shows average accuracy and cost. For all main methods (oracle, 2-model cascades, and task cascades), results are averaged over three trials; variants are single-trial. Baseline costs are shown in USD. Other costs are reported as a multiple of the corresponding 2-Model Cascade variant (plain or +G). The "Avg. Cost" column averages over workloads where the method met the 90% target. Bolded entries mark the lowest cost among methods on each workload.

Method	AGNEWS	COURT	ENRON	FEVER	GAMES	LEGAL	PUBMED	WIKI_TALK	Avg. Cost
`Oracle Only`	\$0.45	\$11.05	\$6.34	\$13.01	\$3.13	\$10.20	\$8.36	\$2.94	\$6.94
`2-Model Cascade`	94.1% | \$0.03	89.4% | \$2.81	89.1% | \$0.61	94.4% | \$0.80	89.8% | \$0.78	91.5% | \$3.23	92.5% | \$0.56	91.4% | \$0.18	\$1.13
`2-Model Cascades (+G)`	94.7% | \$0.04	93.0% | \$4.06	96.2% | \$1.56	96.0% | \$2.22	95.2% | \$1.29	94.8% | \$4.12	93.6% | \$0.83	95.6% | \$0.49	\$1.83
`Task Cascades`	95.3% | 0.66×	88.0% | 0.72×	95.5% | 0.11×	90.6% | 0.54×	92.0% | 1.09×	91.3% | 0.27×	91.6% | 0.84×	91.5% | 0.64×	0.59×
`Task Cascades (+G)`	95.4% | 0.71×	90.8% | **0.53×**	95.6% | **0.09×**	92.4% | 0.44×	89.2% | **0.49×**	91.2% | 0.26×	93.5% | 1.23×	92.3% | **0.37×**	**0.52×**
`Task Cascades (Lite)`	94.1% | 0.78×	88.6% | 0.76×	95.5% | 0.14×	90.6% | 0.63×	90.2% | 0.70×	90.2% | 0.41×	91.4% | 0.86×	89.9% | 0.68×	0.62×
`No Surrogates`	95.1% | 1.43×	94.5% | 1.25×	98.0% | 0.49×	92.5% | 0.86×	90.5% | 1.16×	90.0% | 0.95×	91.9% | 0.79×	96.0% | 2.77×	1.21×
`Single-Iteration`	92.4% | **0.56×**	93.8% | 1.21×	95.9% | 0.15×	90.0% | **0.30×**	89.2% | 1.21×	92.8% | 1.04×	93.4% | **0.76×**	90.6% | 0.61×	0.66×
`No Filtering`	93.9% | 0.74×	90.0% | 1.10×	98.5% | 0.96×	95.5% | 1.82×	98.4% | 1.84×	96.1% | 0.93×	95.1% | 1.47×	98.4% | 3.51×	1.55×
`Naive RAG Filter`	95.1% | 0.95×	90.9% | 0.94×	97.2% | 0.30×	90.9% | 0.42×	85.1% | 0.61×	88.9% | **0.12×**	90.5% | 0.88×	89.9% | 0.44×	0.65×
`Selectivity Ordering`	94.2% | 1.20×	90.8% | 2.33×	98.8% | 5.50×	93.1% | 5.08×	90.1% | 2.32×	92.8% | 1.86×	93.8% | 8.44×	96.9% | 8.81×	4.44×
`Restructure (Top-25%)`	96.5% | 2.67×	93.9% | 1.22×	99.4% | 0.85×	90.5% | 0.37×	97.1% | 1.70×	95.3% | 1.78×	92.6% | 2.38×	98.1% | 3.49×	1.81×
`RAG + NoSur`	96.0% | 1.83×	92.1% | 1.10×	98.2% | 1.73×	91.2% | 0.54×	87.9% | 1.09×	90.0% | 0.77×	90.5% | 0.94×	93.4% | 1.21×	1.16×

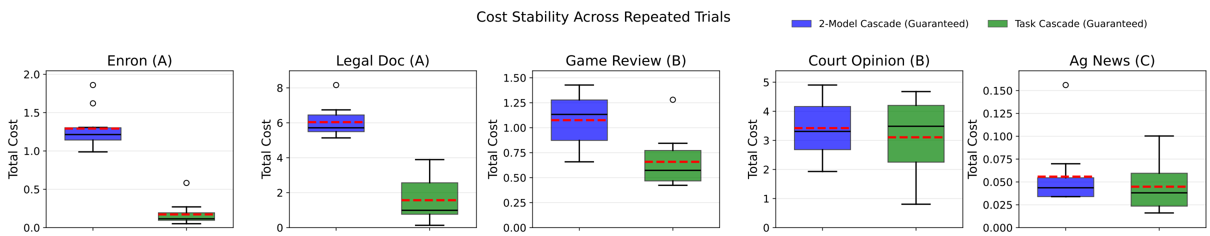

Figure 4: Cost stability of `2-Model Cascade (+ Guarantees)` and `Task Cascades (+ Guarantees)` across 10 independent runs for five representative workloads. Each box shows the distribution of total inference cost; red dashed lines indicate the median. Both methods exhibit high variance.

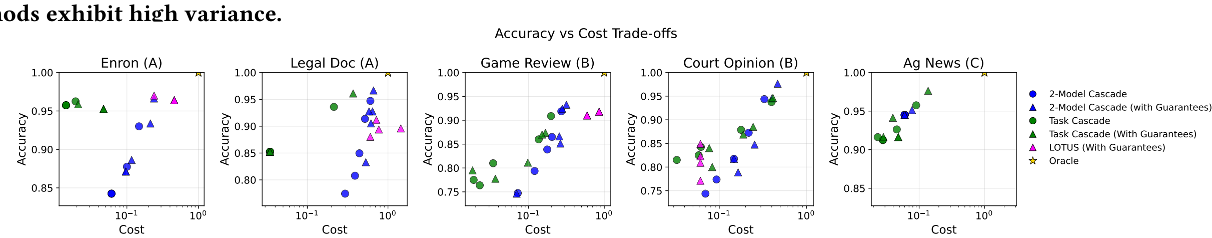

Figure 5: Accuracy vs. cost trade-offs as the target accuracy varies 75% to 95% (in increments of 5%). Task cascades (green) dominate the Pareto frontier on easy workloads and provide robust gains or new operating points on harder tasks. On simple workloads like AGNEWS, cost improvements appear mainly at lower targets, where the baselines cannot match. `LOTUS (+ Guarantees)` (pink) only supports binary classification workloads, so we exclude AGNEWS.

runs, given the inherent stochasticity of LLM-based surrogate discovery. To provide a representative analysis, while managing compute and cost constraints, we select five workloads spanning a range of document lengths and task complexities. We organize these into three groups based on observed outcomes in Table 3: Group A (ENRON and LEGAL), where task cascades deliver the largest cost reductions; Group B (GAMES and COURT), where it is most difficult to find cheap and accurate cascades; and Group C (AGNEWS), where the proxy model alone is already highly accurate and documents are short, impeding savings from document pruning techniques.

We next evaluate the consistency of task cascade outcomes across repeated runs, given the inherent stochasticity of LLM-based surrogate discovery. To provide a representative analysis while managing compute and cost constraints, we select five workloads spanning a range of document lengths and task complexities. We group them by observed difficulty in Table 3: Group A (ENRON, LEGAL) where task cascades achieve the largest cost reductions; Group B (GAMES, COURT) where it is hardest to find cheap, accurate cascades; and Group C (AGNEWS) where documents are short and the proxy model is already highly accurate. As shown in Figure 4, we evaluate both **`2-Model Cascade (+ Guarantees)`** and **`Task Cascades (+ Guarantees)`** across ten independent runs per workload. On Group A workloads, **`Task Cascades (+ Guarantees)`** reduces average cost by 86.5% on ENRON and 74.1% on LEGAL, and even the *most expensive* task cascade run remains *cheaper than the cheapest* 2-model cascade. On Group B workloads, both methods show wider variance due to increased task difficulty, but the *75th percentile* task cascade cost is still *lower than the 25th percentile* of the baseline cost. On Group C (AGNEWS), where the baseline is already near-optimal, **`Task Cascades (+ Guarantees)`** match its cost and variance.

Both methods exhibit notable run-to-run variance. **`2-Model Cascade (+ Guarantees)`** also shows high variability—likely because the small development set size and data-splitting procedure for threshold adjustment introduce uncertainty in cascade construction. `Task Cascades (+ Guarantees)` consistently achieves lower mean and median cost across all workloads, demonstrating that variability does not undermine its advantage.

7.2.4 Q4: Optimization Costs. So far, we have focused on inference costs, as these dominate in production-scale deployments, where offline optimization costs are offset by online gains. To examine when this investment "pays off", we report optimization costs, break-even points (when total cost becomes lower than running `Oracle Only`), and estimated costs for processing 1 million documents (selected to showcase the impact at scale). Full details are provided in Appendix E, described in Table 4.

At 1M documents, optimization costs become negligible and task cascades provide substantial savings. `Task Cascades (Lite)`—our lightweight variant, described in Section 7.1—provides cost savings over `2-Model Cascade` on all workloads, with median savings of 27% and up to 86% on ENRON. `Task Cascades` provides savings on 7 of 8 workloads (median 29%, up to 87% on ENRON); on GAMES, where `Task Cascades` does not provide savings, `2-Model Cascade` fails to meet the 90% accuracy target (Table 3).

These savings are achieved even though `Task Cascades` incurs higher optimization costs than `2-Model Cascade` (7.9×–27.0× across workloads, mean 11.4×) and takes longer to build (~ 14× optimization latency). Our lightweight `Task Cascades (Lite)` variant costs 2.2×–4.7× the `2-Model Cascade` baseline (mean 2.9×) while preserving 95% of inference savings. Optimization cost is recovered after processing 2,986 documents on average for `Task Cascades`, while `Task Cascades (Lite)` requires only 678—modest thresholds relative to our workload sizes (median 129k documents, as in Table 2). Inference latencies are within the same order of magnitude (Table 5 in Appendix E).

In summary, while task cascades require higher upfront optimization investment, this cost quickly becomes negligible at realistic scales, demonstrating that task cascades are practical and cost-effective for production deployments.

7.3 Insights from Analyzing Task Cascades

Choosing Variants of the Task Cascade Approach. The `Task Cascades` approach provides the most consistent cost reductions across workloads, making it a reliable default. Simpler variants may be preferable in specific scenarios: `Naive RAG Filter` when engineering effort is limited and documents have clearly identifiable relevant sections; `No Surrogates` when the original operation is not very difficult and the proxy can accurately perform it; and variants without filtering when documents are very short (e.g., AGNEWS). The `Single-Iteration` variant is particularly promising, achieving strong performance with reduced offline cost and likely to improve as LLM agents improve in their ability to reason and handle complex tasks. While individual variants occasionally outperform the full approach, `Task Cascades` is 20% cheaper than the next best variant on average, making it the recommended starting point.

Parameter Selection Guidance. Task cascades depend on three hyperparameters: the number of surrogate operations per iteration (n_s), refinement iterations (n_a), and document fraction sets ($\mathcal{F}$). We conduct sensitivity analysis on two representative workloads, varying $n_s \in \{3, 5, 10\}$, $n_a \in \{1, 2, 3\}$, and testing different $\mathcal{F}$ configurations (detailed results in Appendix D). Task cascades consistently outperform baselines across all configurations, demonstrating robustness to parameter choices. Based on these experiments, we recommend: $n_s \geq 3$, $n_a \geq 1$, and document fractions spanning from small (0.1 or 0.25) to full (1.0). We find that n_s has a stronger effect on cost reduction than n_a, so practitioners should prioritize exploring diverse surrogates over extensive refinement.

8 RELATED WORK

We cover three areas: LLM-powered data processing, cost-efficient LLM execution, and query rewriting with LLMs.

LLM-Powered Data Processing. Modern data systems increasingly integrate LLMs as first-class operators for natural language extraction and transformation. Industrial systems like Databricks, DuckDB, Snowflake, and Google's AlloyDB support LLM-powered *filter* and *map* functions in SQL [16, 21, 55, 67]. LOTUS [49], ThalamusDB [30], and Aryn [1] provide pandas-based, SQL, and Spark-like interfaces respectively. Palimpzest [41], DocETL [53], and AOP [65] offer custom DSLs for LLM-powered data processing. However, these systems typically invoke LLMs on each document independently, resulting in high inference costs. Some systems focus on supporting LLM-powered data processing for particular query or data modalities [3, 40, 44, 61].

Cost Optimization for LLM Inference. Strategies for reducing LLM inference costs include cost-based optimizers, specialized models, caching, ensembling, and model cascades. ABACUS [52] introduces a Cascades-style optimizer [23] over different "physical" implementations of common LLM-powered operators, while ELEET [62] replaces LLM operators with trainable models. Other work reorders LLM calls for KV-cache reuse [43]. Some methods profile multiple LLMs and aggregate their outputs to reduce cost. For example, ThriftLLM [27] and LLMBlender [28], instead of using only the output of the last processed stage as in model cascades, ensemble predictions from all evaluated models. However, in the primary setting we study—where cascade models differ substantially in cost and quality—there is no additional value to be gained by ensembling the oracle's output (when available) with that of the proxy. SpareLLM [31] profiles a number of LLMs for a given family of tasks, but ultimately selects a single model to handle the remaining tasks, which in our case boils down to a cascade with a single step, either a proxy model or an oracle, for all items, independent of confidence. Our work builds on model cascades [2, 5, 9, 33–35, 45, 49, 64, 70], generalizing them by varying not only the model but also the operation and document fraction at each stage.

Query Rewriting with LLMs. LLM-powered query rewriting has proven effective in retrieval, question-answering [4, 10, 46, 50], and traditional query optimization [15, 39, 57, 72]. DocETL [53] decomposes complex tasks into logically equivalent subtasks for accuracy. In contrast, we rewrite tasks for cheaper, possibly-incorrect operations, dramatically expanding the rewrite space. Task cascades may not resemble logical decompositions—the same operation can appear multiple times at different document fractions.

Our work also builds on approximate query processing (AQP) [7, 19]. While tailored for unstructured data and LLMs, our components mirror classical AQP: surrogate operations resemble approximate predicates [54], document fractions leverage sampling, and we use concentration bounds for accuracy guarantees [8, 35].

9 CONCLUSION

We introduce *task cascades*, a generalization of the model cascades framework for efficient LLM-powered unstructured text processing. Task cascades vary the model, operation, and document fraction at each stage to minimize inference cost while meeting accuracy targets. Across eight real-world workloads, task cascades reduce inference cost by 48.5% on average compared to model cascades (with comparable guarantees) and by 86.2% relative to oracle-only inference, with ablations confirming the importance of all components of our approach. Overall, task cascades provide a practical and extensible solution for scalable unstructured data analysis.

REFERENCES

[1] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A. Shah, Benjamin Sowell, Dan Tecuci, Vinayak Thapliyal, and Matt Welsh. 2025. The Design of an LLM-powered Unstructured Analytics System. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. https://mail.vldb.org/cidrdb/papers/2025/p13-anderson.pdf

[2] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. 2019. Physical representation-based predicate optimization for a visual analytics database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1466–1477.

[3] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proceedings of the VLDB Endowment* 17, 2 (2023), 92–105.

[4] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–28.

[5] Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2022. Figo: Fine-grained query optimization in video analytics. In *Proceedings of the 2022 International conference on management of data*. 559–572.

[6] Jonathan P Chang, Caleb Chiam, Liye Fu, Andrew Wang, Justine Zhang, and Cristian Danescu-Niculescu-Mizil. 2020. ConvoKit: A Toolkit for the Analysis of Conversations. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 57–60.

[7] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 511–519.

[8] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1998. Random sampling for histogram construction: How much is enough? *ACM SIGMOD Record* 27, 2 (1998), 436–447.

[9] Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Transactions on Machine Learning Research* (2024).

[10] Peter Baile Chen, Yi Zhang, Mike Cafarella, and Dan Roth. [n.d.]. Can we Retrieve Everything All at Once? ARM: An Alignment-Oriented LLM-based Retrieval Method. In *The 4th Table Representation Learning Workshop at ACL 2025*.

[11] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 615–621. https://doi.org/10.18653/v1/N18-2097

[12] Cristian Danescu-Niculescu-Mizil, Lillian Lee, Bo Pang, and Jon Kleinberg. 2012. Echoes of power: Language effects and power differences in social interaction. In *Proceedings of the 21st international conference on World Wide Web*. 699–708.

[13] Gianna M Del Corso, Antonio Gulli, and Francesco Romani. 2005. Ranking a stream of news. In *Proceedings of the 14th international conference on World Wide Web*. 97–106.

[14] Shrey Desai and Greg Durrett. 2020. Calibration of Pre-trained Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 295–302.

[15] Sriram Dharwada, Himanshu Devrani, Jayant Haritsa, and Harish Doraiswamy. 2025. Query rewriting via llms. *arXiv preprint arXiv:2502.12918* (2025).

[16] Till Döhmen. 2024. Introducing the prompt() Function: Use the Power of LLMs with SQL! urlhttps://motherduck.com/blog/sql-llm-prompt-function-gpt-models/. Accessed: 2025-06-22.

[17] Yu Fan, Jingwei Ni, Jakob Merane, Etienne Salimbeni, Yang Tian, Yoan Hermstrüwer, Yinya Huang, Mubashara Akhtar, Florian Geering, Oliver Dreyer, et al. 2025. LEXam: Benchmarking Legal Reasoning on 340 Law Exams. *arXiv preprint arXiv:2505.12864* (2025).

[18] Uriel Feige, László Lovász, and Prasad Tetali. 2004. Approximating min sum set cover. *Algorithmica* 40 (2004), 219–234.

[19] Minos N Garofalakis and Phillip B Gibbons. 2001. Approximate Query Processing: Taming the TeraBytes.. In *VLDB*, Vol. 10. 645927–672356.

[20] Google Cloud. 2025. Batch prediction with Gemini. https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/batch-prediction-gemini. Last updated 2025-10-15; accessed: 2025-10-16.

[21] Google Cloud. 2025. Perform intelligent SQL queries using AlloyDB AI query engine. https://cloud.google.com/alloydb/docs/ai/evaluate-semantic-queries-ai-operators. Accessed: 2025-06-22; Last updated: 2025-06-11.

[22] gqfiddler. 2022. SCOTUS Opinions. https://www.kaggle.com/datasets/gqfiddler/scotus-opinions. Accessed: 2025-07-16.

[23] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data(base) Engineering Bulletin* 18 (1995), 19–29. https://api.semanticscholar.org/CorpusID:260706023

[24] Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. [n.d.]. Language Model Cascades: Token-Level Uncertainty And Beyond. In *The Twelfth International Conference on Learning Representations*.

[25] Joseph M. Hellerstein and Michael Stonebraker. 1993. Predicate migration: optimizing queries with expensive predicates. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., USA) *(SIGMOD '93)*. Association for Computing Machinery, New York, NY, USA, 267–276. https://doi.org/10.1145/170035.170078

[26] Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. 2021. CUAD: An Expert-Annotated NLP Dataset for Legal Contract Review. *NeurIPS* (2021).

[27] Keke Huang, Yimin Shi, Dujian Ding, Yifei Li, Yang Fei, Laks Lakshmanan, and Xiaokui Xiao. 2025. ThriftLLM: On Cost-Effective Selection of Large Language Models for Classification Queries. *arXiv preprint arXiv:2501.04901* (2025).

[28] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 14165–14178.

[29] Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. 2021. How can we know when language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics* 9 (2021), 962–977.

[30] Saehan Jo and Immanuel Trummer. 2024. Thalamusdb: Approximate query processing on multi-modal data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.

[31] Saehan Jo and Immanuel Trummer. 2025. SpareLLM: Automatically Selecting Task-Specific Minimum-Cost Large Language Models under Equivalence Constraint. *Proc. ACM Manag. Data* 3, 3, Article 219 (June 2025), 26 pages. https://doi.org/10.1145/3725356

[32] Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221* (2022).

[33] Daniel Kang, Peter Bailis, and Matei Zaharia. [n.d.]. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proceedings of the VLDB Endowment* 13, 4 ([n. d.]).

[34] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment* 10, 11 (2017).

[35] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate selection with guarantees using proxies. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1990–2003.

[36] Daniel Kang and John Guibas. 2021. Accelerating Approximate Aggregation Queries with Expensive Predicates. *Proc. VLDB Endow.* 14 (2021), 2341–2354. https://api.semanticscholar.org/CorpusID:237012342

[37] Daniel Kang, John Guibas, Peter D Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. Tasti: Semantic indexes for machine learning-based queries over unstructured data. In *Proceedings of the 2022 International Conference on Management of Data*. 1934–1947.

[38] Bryan Klimt and Yiming Yang. 2004. The enron corpus: A new dataset for email classification research. In *European conference on machine learning*. Springer, 217–226.

[39] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-Based Rewrite System for Boosting Query Efficiency. *Proceedings of the VLDB Endowment* 18, 1 (2024),

53–65.
[40] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeighami, Aditya G Parameswaran, and Eugene Wu. 2025. Querying Templatized Document Collections with Large Language Models. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2422–2435.
[41] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*. 2.
[42] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.
[43] Shu Liu, Asim Biswal, Amog Kamsetty, Audrey Cheng, Luis Gaspar Schroeder, Liana Patel, Shiyi Cao, Xiangxi Mo, Ion Stoica, Joseph E Gonzalez, et al. 2025. Optimizing LLM Queries in Relational Data Analytics Workloads. In *Eighth Conference on Machine Learning and Systems*.
[44] Duo Lu, Siming Feng, Jonathan Zhou, Franco Solleza, Malte Schwarzkopf, and Uğur Çetintemel. 2025. VectraFlow: Integrating Vectors into Stream Processing. CIDR.
[45] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating machine learning inference with probabilistic predicates. In *Proceedings of the 2018 International Conference on Management of Data*. 1493–1508.
[46] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 5303–5315.
[47] Matteo Magnini, Gianluca Aguzzi, and Sara Montagna. 2025. Open-source small language models for personal medical assistant chatbots. *Intelligence-Based Medicine* 11 (2025), 100197.
[48] OpenAI. 2025. Batch requests guide. https://platform.openai.com/docs/guides/batch. Accessed: 2025-10-16.
[49] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2024. Semantic Operators: A Declarative Model for Rich, AI-based Data Processing. *arXiv preprint arXiv:2407.11418* (2024).
[50] Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Derong Xu, Tong Xu, and Enhong Chen. 2024. Large language model based long-tail query rewriting in taobao search. In *Companion Proceedings of the ACM Web Conference 2024*. 20–28.
[51] Matthew Russo, Tatsunori Hashimoto, Daniel Kang, Yi Sun, and Matei Zaharia. 2023. Accelerating Aggregation Queries on Unstructured Streams of Data. *Proceedings of the VLDB Endowment* 16, 11 (2023), 2897–2910.
[52] Matthew Russo, Sivaprasad Sudhir, Gerardo Vitagliano, Chunwei Liu, Tim Kraska, Samuel Madden, and Michael Cafarella. 2025. Abacus: A Cost-Based Optimizer for Semantic Operator Systems. *arXiv preprint arXiv:2505.14661* (2025).
[53] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. 2024. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *arXiv preprint arXiv:2410.12189* (2024).
[54] N. Shivakumar, H. Garcia-Molina, and C. Chekuri. 1998. Filtering with approximate predicates. In *24rd International Conference on Very Large Data Bases (VLDB 1998)*. http://ilpubs.stanford.edu:8090/351/
[55] Snowflake. 2025. Introducing Cortex AISQL: Reimagining SQL into AI Query Language for Multimodal Data. https://www.snowflake.com/en/blog/ai-sql-query-language/. Accessed: July 15, 2025.
[56] Antoni Sobkowicz and Wojciech Stokowiec. 2016. Steam review dataset-new, large scale sentiment dataset. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016) Workshop Emotion and Sentiment Analysis*. 55–58.
[57] Yuyang Song, Hanxu Yan, Jiale Lao, Yibo Wang, Yufei Li, Yuanchun Zhou, Jianguo Wang, and Mingjie Tang. 2025. QUITE: A Query Rewrite System Beyond Rules with LLM Agents. *arXiv preprint arXiv:2506.07675* (2025).
[58] Sijun Tan, Xiuyu Li, Shishir G Patil, Ziyang Wu, Tianjun Zhang, Kurt Keutzer, Joseph Gonzalez, and Raluca Popa. 2024. LLoCO: Learning Long Contexts Offline. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 17605–17621.
[59] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a Large-scale Dataset for Fact Extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 809–819.
[60] u/ICausedAnOutage and others. 2023. *The response of the azure OpenAI API is much slower than that of ChatGPT official.* https://www.reddit.com/r/AZURE/comments/1c1n48j/the_response_of_the_azure_openai_api_is_much/ Reddit post, accessed 2025-10-16.
[61] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, CA, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p14-urban.pdf
[62] Matthias Urban and Carsten Binnig. 2024. Eleet: Efficient learned query execution over text and tables. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4867–4880.
[63] Vals AI, Inc. 2024. Vals AI. https://www.vals.ai/. Accessed: 2025-06-22.
[64] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, Vol. 1. Ieee, I–I.
[65] Jiayi Wang and Guoliang Li. 2025. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. CIDR.
[66] Ian Waudby-Smith and Aaditya Ramdas. 2024. Estimating means of bounded random variables by betting. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 86, 1 (2024), 1–27.
[67] Patrick Wendell, Eric Peter, Nicolas Pelaez, Jianwei Xie, Vinny Vijeyakumaar, Linhong Liu, and Shitao Li. 2023. Introducing AI Functions: Integrating Large Language Models with Databricks SQL. https://www.databricks.com/blog/2023/04/18/introducing-ai-functions-integrating-large-language-models-databricks-sql.html. Accessed: 2025-06-22.
[68] Xinyi Wu, Yifei Wang, Stefanie Jegelka, and Ali Jadbabaie. 2025. On the Emergence of Position Bias in Transformers. *arXiv preprint arXiv:2502.01951* (2025).
[69] Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Retrieval meets long context large language models. In *The Twelfth International Conference on Learning Representations*.
[70] Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2024. Large Language Model Cascades with Mixture of Thought Representations for Cost-Efficient Reasoning. In *The Twelfth International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=6okaSfANzh
[71] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.
[72] Xuanhe Zhou, Guoliang Li, Jianming Wu, Jiesi Liu, Zhaoyan Sun, and Xinning Zhang. 2023. A learned query rewrite system. *Proceedings of the VLDB Endowment* 16, 12 (2023), 4110–4113.

A CASCADE ASSEMBLY AND STATISTICAL GUARANTEES FOR CASCADE ACCURACY

A.1 Greedy Cascade Assembly Algorithm

In this section, we provide the pseudocode for the greedy cascade assembly algorithm referenced in Section 3.2.2. This algorithm takes as input a set of eligible tasks (filtered and assigned thresholds via Algorithm 2) and greedily constructs an ordered cascade by iteratively selecting the task that maximally reduces inference cost while maintaining per-task accuracy requirements.

Algorithm 4: Greedy Cascade Assembly (detailed)

Input: Eligible tasks $\mathcal{T}_{\text{eligible}}$ with thresholds, development set D_{dev}, accuracy target α
Output: Ordered cascade π

$\pi \leftarrow \emptyset$;
while *true* **do**
    `best_task` $\leftarrow$ None;   `best_cost` $\leftarrow$ cost of π;
    foreach *unused task T in* $\mathcal{T}_{\text{eligible}}$ **do**
        $\pi' \leftarrow \pi + [T]$;   `// Append` T `to end of cascade`
        Execute π' on D_{dev};
        if *for every task in π', accuracy on the documents it classifies (i.e., with confidence above threshold) $\geq \alpha$ and cost(π') <* `best_cost` **then**
            `best_task` $\leftarrow T$;   `best_cost` $\leftarrow$ cost(π');
        end
    end
    if `best_task` *is None* **then**
        break
    end
    $\pi \leftarrow \pi +$ [`best_task`] ;   `// Update current cascade`
end
return π;

A.2 Estimator Details and Proof of Theorem 3.2

Estimation function. To meet the target accuracy, the function $\mathcal{E}$ needs to correctly estimate, based on the sampled set D_V, whether a threshold set meets the accuracy target or not. More formally, let π_i be the model cascade when using the threshold set $\mathbf{t}_i$, and let $\pi(x_j)$ for $(x_j, y_i) \in D_V$ be the output of the model cascade on input sample x whose ground-truth answer is y. Define $X^i_j = \mathbb{I}[\pi_i(x_j) = y_j]$. Note that $\mathbb{E}(X^i_j = 1) = \text{Acc}_D(\pi_i)$, so that the goal of $\mathcal{E}$ is to estimate, using a set of observed i.i.d Bernoulli random variables $X^i = \{X^i_j; \forall j\}$ if their mean is more than the target T or not. This can be achieved by an application of common concentration bounds such as Hoeffding's inequality. We use the recent results by Waudby-Smith and Ramdas [66] that provides tighter bounds that Hoeffding's inequality as shown in [66]. The results of [66] takes both means and *variances* of the observations into account to estimate their mean, thus yielding tighter bounds. The following is a restatement of the Theorem 3.2 by [66], defining our function $\mathcal{E}$ and showing that it can estimate whether the mean of observation is more than T or not with high probability.

Lemma A.1 (Corollary to Theorem 3.2 by [66]). *Consider the i-th cascade threshold set $\mathbf{t}_i$ and the corresponding Bernoulli random variables X^i with $|X^i| = k$. If $\text{Acc}_D(\mathbf{t}_i) < T$ and for a confidence parameter $\alpha \in [0, 1]$,*

$$\mathbb{P}(\mathcal{E}(\mathbf{t}, D_V) = True) \leq \alpha, \quad \textit{where} \tag{1}$$

$$\mathcal{E}(\mathbf{t}_i, D_V) = \mathbb{I}[\exists j \in [k] \; s.t. \mathcal{K}(T, X^i[: j]) \geq \frac{1}{\alpha}]. \tag{2}$$

$\mathcal{K}(T, X)$ for a set of i random variables $X = \{X_1, ..., X_i\}$ is defined as

$$\mathcal{K}(T, X) = \Pi^i_{j=1}(1 + \min(\lambda_j, \frac{3}{4T}) \times (X_j - T)), \tag{3}$$

$$\lambda_i = \sqrt{\frac{2\log(2/\delta)}{i\log(i+1)\hat{\sigma}^2_{i-1}}}, \; \hat{\sigma}^2_i = \frac{1/4 + \sum^i_{j=1}(X_j - \hat{\mu}_j)^2}{i+1}, \; \hat{\mu}_i = \frac{1/2 + \sum^i_{j=1} X_j}{i+1}.$$

Proof of Theorem 3.2. Lemma A.1 shows that if the threshold set at the i-th iteration does not meet the target, then the probability that the estimation function returns True is bounded by α. We use this result to show the total probability that Algorithm 3 returns a threshold that doesn't meet the target is also bounded by α. To do so, let $\mathbf{t}_{i^*}$ be the threshold set with the lowest i^* that does not meet the target. Note that Algorithm 3 returns a threshold up to the first threshold that it estimates does not meet the target. Thus, if $\mathcal{E}(\mathbf{t}_{i^*}, D_V) = False$, then Algorithm 3 returns a threshold set that meets the target accuracy (or returns not found). As such, Algorithm 3 returns a threshold that fails to meet the target only if $\mathcal{E}(\mathbf{t}_{i^*}, D_V) = True$. The probability of the latter is bounded by α according to Lemma A.1, so the probability of threshold selected by Algorithm 3 not meeting the target is also bounded by α. □

A.3 Threshold Adjustment Procedure

Algorithm 5: ThresholdShift

Input: Candidate threshold lists $\mathbf{p}^c_T$ for each task and class; validation set D_V; estimator $\mathcal{E}$ (constructed to certify accuracy α at failure probability δ)
Output: Adjusted thresholds $\mathbf{t}^*$ such that $\Pr[\text{Acc}(\pi^*) < \alpha] \leq \delta$

`best_t` $\leftarrow$ thresholds with maximal shift (most conservative);
for $s = s_{\max}$ ***down to*** 0 **do**
    foreach *class c for each task T* **do**
        Initialize $\mathbf{t} \leftarrow \emptyset$;
        if $s <$ *length of* $\mathbf{p}^c_T$ **then**
            Set $\tau^c = p_s$;   `//` $\mathbf{p}^c_T = (p_0, p_1, \ldots, p_{s_{\max}})$
        end
        else
            Set $\tau^c = \infty$;   `// Disable class if shift exceeds available values`
        end
        Add τ^c to $\mathbf{t}$ for (T, c);
    end
    if $\mathcal{E}(\mathbf{t}, D_V)$ *is* `True` **then**
        `best_t` $\leftarrow \mathbf{t}$;
    end
    else
        break
    end
end
return `best_t`;

The final step of cascade construction (see Section 3.2.3) adjusts class thresholds to ensure the cascade achieves the target accuracy α on unseen data with probability at least $1 - \delta$. For each class c in each task, we construct a shift list of candidate thresholds $p_1 < p_2 < \cdots < p_k$, representing increasingly conservative cutoffs above the original threshold τ^c. For each shift value s from k down to 0, we set the threshold for class c to p_s for $s > 0$, and to τ^c for

$s = 0$; if no such threshold exists for s, we set $\tau^c = \infty$, disabling predictions of that class at that task. At each candidate threshold set $\mathbf{t}_s$, we evaluate the cascade on the validation set D_V and apply the estimator $\mathcal{E}$ to certify whether the accuracy guarantee is satisfied. We return the least conservative (smallest s) threshold set that meets the guarantee. If no threshold set passes, we revert to the oracle-only cascade (this was never triggered in our experiments).

The full algorithm is presented in Algorithm 5.

Algorithm 6: Agentic Loop for Surrogate Cascade Refinement

Input: Original operation o_{orig}, development set D_{dev}, models $\mathcal{M}$, document fractions $\mathcal{F}$, accuracy target α, failure probability δ, rounds n_a, surrogates per round n_s
Output: Final cascade π^*

// Initialize candidate tasks
$\mathcal{T} \leftarrow \{(m, o_{\text{orig}}, f) : m \in \mathcal{M}, f \in \mathcal{F}\}$
for $r = 1$ **to** n_a **do**
  // Assemble best cascade using CascadeAssembly (Algorithm 4)
  $\pi_{\text{current}} \leftarrow \text{CascadeAssembly}(\mathcal{T}, D_{\text{dev}}, \alpha, \delta)$
  // Collect feedback and failure cases
  foreach *task T in $\mathcal{T}$* **do**
    Determine if T is selected in π_{current}; record its coverage (number of documents classified) and up to 10 hard misclassified examples near threshold (confidence just above τ).
  end
  Identify documents not classified by any non-oracle task (i.e., routed to the oracle); extract minimal supporting spans using the oracle for prompt brevity.
  // Elicit new surrogates from agent
  Provide agent with: (i) user operation, (ii) oracle-extracted failure cases, (iii) per-task statistics, (iv) explicit surrogate generation instructions.
  Receive n_s new surrogate operations.
  // Expand candidate set
  foreach *new surrogate o_{sur}* **do**
    Add (m, o_{sur}, f) to $\mathcal{T}$ for all $m \in \mathcal{M}, f \in \mathcal{F}$
  end
end
return *CascadeAssembly*$(\mathcal{T}, D_{\text{dev}}, \alpha, \delta)$

B SURROGATE OPERATION GENERATION: AGENTIC LOOP

Algorithm 6 describes the procedure for surrogate operation generation. At each iteration, we provide the LLM agent with a prompt structured as in Figure 6.

The prompt in Figure 6 is used at every iteration of the agentic loop, updated each time with new failure cases and task statistics. To help the agent generate useful surrogate operations, we provide a short list of concrete detection strategies, such as entity detection, event detection, relationship extraction, and context cues, which are easily translated into simple classification rules. We also include an in-context example showing how to turn these strategies into specific instructions. This example is only for demonstration and is not used in our actual experiments. The complete prompt template is available in our codebase.

While these detection strategies help guide the agent toward productive proposals, they are not strictly necessary—a good agent may discover effective surrogates even without this guidance. A systematic study of prompt engineering for surrogate generation, including the potential for agent fine-tuning, is left to future work.

C EXAMPLES OF SURROGATE OPERATIONS

To understand how task cascades achieve cost savings, we examined the surrogate instructions produced by our agent and identified four main types. **Keyword or phrase-based surrogates** rely on the presence of predictive terms or domain jargon; for example, in WIKI_TALK, a surrogate checked for "WP:3RR" or "three-revert rule," and in GAMES, for comparative phrases like "better than" or "prefer over." **Class-specific surrogates** target features associated with a single class: for AGNEWS, one surrogate detected references to corporate mergers or financial metrics to identify business articles; for PUBMED, another flagged documents as "research" if they contained phrases like "in vitro" or "in vivo." **Semantic pattern-based surrogates** capture higher-level cues correlated with the target; for example, in ENRON, a surrogate looked for a formal signature with title and contact information to identify executive emails. Fourth, **sequential decomposition surrogates** break the original operation into simpler steps; in GAMES, one surrogate checked only whether any other game was mentioned—a necessary prerequisite for assessing comparative sentiment. In some cases, the agent found no helpful surrogate; for example, in FEVER, the best cascade simply reapplied the original user instruction to a subset of the document. The agent also sometimes reused the same surrogate at multiple document fractions, such as in LEGAL, where it checked for "agreement not to sue or challenge intellectual property rights" at both 0.1 and 1.0 fractions.

D PARAMETER SENSITIVITY ANALYSIS

Task cascades depend on three main hyperparameters: (1) the number of surrogate operations proposed per iteration (n_s), (2) the number of agentic refinement iterations (n_a), and (3) the set of document fractions to consider ($\mathcal{F}$). To evaluate robustness to these choices, we conduct a sensitivity analysis on two representative workloads: ENRON (where task cascades achieve large gains) and GAMES (where gains are modest). For each workload, we vary one parameter at a time while holding others at their default values: $n_s \in \{3, 5, 10\}$ with $n_a = 3$ fixed, $n_a \in \{1, 2\}$ with $n_s = 5$ fixed, and $\mathcal{F} \in \{\{0.25, 1.0\}, \{0.25, 0.5, 1.0\}\}$ with $n_s = 5, n_a = 1$. All experiments use a 90% accuracy target and report costs relative to the `2-Model Cascade` baseline.

Figure 7 shows the results. `Task Cascades` consistently outperform baselines across all parameter configurations on both workloads. On ENRON, all settings achieve substantial cost savings over the `2-Model Cascade` baseline while meeting the accuracy target. On GAMES, where `Task Cascades` shows only modest improvements over `2-Model Cascade`, parameter variations remain stable, demonstrating robustness even in challenging scenarios.

The sensitivity analysis sheds light on several practical insights. First, n_s (the number of surrogate operations to explore) has a more pronounced effect on cost than n_a (the number of refinement iterations). This suggests it is worth generating multiple diverse surrogate candidates per iteration rather than relying heavily on iterative refinement. Second, all tested document fraction sets yield similar performance, indicating that the approach is robust to this choice. However, it is important to include the full document fraction (1.0) in $\mathcal{F}$, as some tasks or proxy models may benefit from complete context. We recommend spanning a range of fractions—from a

```
Your job is to propose [n_s] simple surrogate operations for the classification task below. Each surrogate must target a different detection
      type from the following list:
- Entity Detection (checks for presence of a specific entity)
- Event Detection (detects a particular event or outcome)
- Relationship Detection (identifies a connection or association)
- Context Detection (determines the broader setting)
- Attribute Detection (checks for a property or attribute)
- Any other type not mentioned above

Each surrogate should be much simpler than the original task, and you must use a unique detection type for each. For a classification task, each
       surrogate's outputs must be a subset of the original task's outputs (if multiclass, may also output -1 for ``none of the above''; if
      binary, output must be True or False).

TASK:
[insert user-defined operation here]

FAILURE CASES:
Here are examples of documents the current cascade fails to classify with any non-oracle task. Only the most relevant text span from each
      document is shown (as extracted by the oracle model):

[insert minimal relevant spans]

TASK STATISTICS:
For each candidate task:
- Selected: [yes/no]
- Coverage: [number of documents classified]
- Hard examples: Up to 10 misclassified documents for this task, showing only the minimal relevant span from each document (as extracted by the
      oracle model).

INSTRUCTIONS:
Propose [n_s] surrogate operations, each corresponding to a different detection type above and distinct from surrogates previously generated.
      Surrogates should be iteratively refined based on the task statistics and failure patterns above.

For each surrogate, provide:

PROMPT: <a concise classification instruction, with allowable outputs matching the original task>
RATIONALE: <what it detects, which detection type it uses, and why it is simpler or complementary to previous surrogates>

EXAMPLE (for a binary task; determining if a medical article describes adverse drug reactions):

TASK:
Does this article describe a negative reaction to a drug? Output True if a negative drug reaction is present, False if not.

Example surrogate operations:

PROMPT: Does the article describe a patient outcome such as rash, nausea, or toxicity? If yes, output True. Otherwise, output False.
RATIONALE: Event Detection: identifies negative outcomes, without requiring explicit attribution to a drug.

PROMPT: Is the article a case report? If yes, output True. Otherwise, output False.
RATIONALE: Context Detection: case reports often highlight adverse reactions in individual patients.

YOUR ANSWER HERE:
```

Figure 6: Prompt template used to elicit surrogates from the agent.

small portion (e.g., 0.1 or 0.25) to the full document—to enable effective document pruning while maintaining flexibility. Overall, these findings suggest that modest parameter values ($n_s \geq 3$, $n_a \geq 1$, and $\mathcal{F}$ spanning from small to full document) are sufficient for strong performance, though users can increase n_s when additional optimization budget is available.

E OPTIMIZATION COST AND LATENCY ANALYSIS

In this section, we provide detailed measurements of the offline optimization costs and latencies for both **`Task Cascades`** and **`2-Model Cascade`**, as well as analysis showing when the upfront optimization investment is recovered through inference savings and total costs at scale.

E.1 Optimization Cost and Break-Even Analysis

Table 4 reports optimization costs, break-even points, and estimated end-to-end costs for processing 1 million documents for each workload. We define the break-even point as the number of inference documents required for the total cost of a method—its one-time optimization cost plus per-document inference—to become lower than simply running the oracle model on every document.

Optimization Cost. As explained in Section 6, **`Task Cascades`** incurs higher optimization costs because it performs two additional

Table 4: Offline optimization costs, break-even analysis, and estimated cost at 1M documents. Left columns (Optimization Cost) report the US dollars required to build each method (`TC`, `TC (Lite)`, `2MC`), given $D_{\text{dev}} = 200$ (except 150 for LEGAL). Middle columns (Break-Even) show how many inference documents must be processed before total cost becomes lower than running the `Oracle Only` on every document. Rightmost columns estimate total end-to-end cost for processing 1 million documents: offline optimization + per-document inference. Parentheses show, for each `TC` variant, its ratio to the `2MC` baseline.

Workload	Optimization Cost ($)			Break-Even (Documents)			Estimated Total Cost @ 1M Docs ($)		
	TC	TC (Lite)	2MC	TC	TC (Lite)	2MC	TC	TC (Lite)	2MC
AGNEWS	$3.32	$0.58	$0.12	6,173.87 (27.0×)	1,080.54 (4.7×)	228.57	$28.07 (0.75×)	$29.83 (0.79×)	$37.62
COURT	$29.51	$7.59	$2.70	2,615.32 (10.0×)	680.85 (2.6×)	262.14	$2,558.51 (0.73×)	$2,677.09 (0.76×)	$3,515.20
ENRON	$13.78	$3.38	$1.15	1,757.40 (10.9×)	431.84 (2.7×)	160.56	$97.65 (0.13×)	$110.13 (0.14×)	$763.65
FEVER	$39.52	$10.27	$3.69	2,513.60 (10.4×)	656.72 (2.7×)	241.77	$579.52 (0.58×)	$640.27 (0.64×)	$1,003.69
GAMES	$10.92	$2.61	$0.87	3,831.92 (12.9×)	808.24 (2.7×)	296.17	$1,073.67 (1.10×)	$685.11 (0.70×)	$975.87
LEGAL	$45.22	$11.89	$4.30	1,745.22 (7.9×)	482.16 (2.2×)	222.09	$1,748.54 (0.28×)	$2,598.41 (0.41×)	$6,312.89
PUBMED	$25.22	$6.44	$2.28	2,557.29 (10.9×)	653.78 (2.8×)	233.85	$613.22 (0.87×)	$608.44 (0.87×)	$702.28
WIKI_TALK	$9.49	$2.23	$0.72	2,687.62 (12.9×)	632.54 (3.0×)	208.70	$153.49 (0.68×)	$155.23 (0.69×)	$225.72

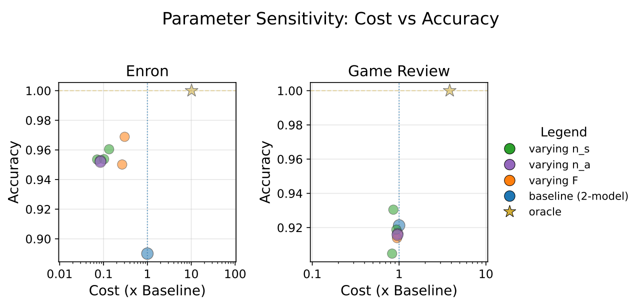

Figure 7: Parameter sensitivity analysis on two workloads representing different performance regimes. Each point shows cost (relative to `2-Model Cascade` baseline) and accuracy for different parameter settings. `Task cascades` demonstrates low variance across all parameter configurations, even on GAMES where baseline performance is strong.

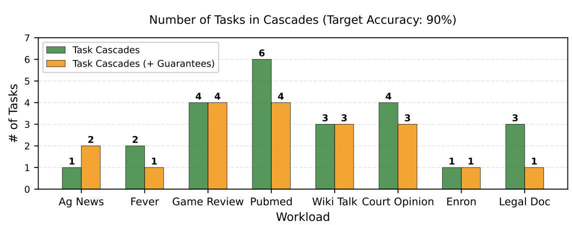

Figure 8: Number of tasks in each cascade for the `Task Cascades` and `Task Cascades (+ Guarantees)` methods at $\alpha = 90\%$.

procedures: *(i)* training a lightweight relevance classifier for document restructuring, and *(ii)* evaluating multiple candidate rewrites of the pipeline on the development set during cascade assembly. In contrast, **`2-Model Cascade`** only evaluates the proxy and oracle models to set confidence thresholds. The magnitude of this cost depends on document length and task complexity: longer documents (FEVER, LEGAL) increase oracle labeling costs, while harder tasks require more surrogate generation rounds.

We also include **`Task Cascades (Lite)`**, a lightweight variant that does not consider candidate tasks that leverage the oracle model, as described in Section 7.1. This configuration reduces optimization

Table 5: Latency measurements (mean ± stdev, in seconds) for optimization and inference (inf.). Due to availability of credits, `2MC`, `TC (Lite)`, and `Oracle Only` used Azure OpenAI APIs, while `TC` used standard OpenAI endpoints (Azure is ∼2.5× slower [60]). Standard deviations are high for all methods, due to API tail latencies. Rightmost column estimates total inference latency for processing 1M documents of API requests, in hours. Estimations represent a lower bound, since they do not account for API tail latencies.

Method	Optimization (s)	Inf. @ ∼1K docs (s)	Est. Inf. @ 1M docs (h)
`2MC`	64.93 ± 57.60	280.64 ± 291.36	3.0 ± 0.1
`TC`	911.45 ± 381.23	201.19 ± 149.03	2.2 ± 0.0
`TC (Lite)`	1253.0 ± 947.3	175.4 ± 90.6	1.9 ± 0.0
`Oracle Only`	–	26.14 ± 56.80	0.3 ± 0.0

cost by roughly 70–80% (2.2×–4.7× the **`2-Model Cascade`** baseline, mean 2.9×) while maintaining similar inference behavior.

Break-Even Analysis. Across workloads, **`Task Cascades`** reaches break-even between 1,745 documents (LEGAL) and 6,174 documents (AGNEWS), with a mean of 2,986. **`Task Cascades (Lite)`** achieves proportionally lower break-even thresholds (480–1,080 documents, mean 678) due to its smaller optimization cost. The ratio between **`Task Cascades`** and **`2-Model Cascade`** break-even points ranges from 7.9× (LEGAL) to 27.0× (AGNEWS), with an overall mean of 11.4×—reflecting the varying magnitude of inference savings.

Costs at Scale. The rightmost columns of Table 4 estimate total end-to-end cost for processing 1 million documents (optimization + inference). At this scale, optimization costs become negligible. **`Task Cascades (Lite)`** provides cost savings over **`2-Model Cascade`** on all 8 workloads, with median savings of $327 per million documents and up to $3,714 on LEGAL. **`Task Cascades`** provides savings on 7 of 8 workloads (all except GAMES), with median savings of $257 and up to $4,564 on LEGAL. On GAMES, **`2-Model Cascade`** fails to meet the 90% accuracy target (Table 3), while both **`Task Cascades`** variants do.

Practical Implications. Most workloads in our evaluation contain tens of thousands to millions of documents (Table 2), making these optimization costs negligible at scale. For example, running on the full ENRON corpus (∼500k documents) requires only 0.35% of documents to reach break-even, while the smaller COURT dataset

(~36k documents) reaches break-even after processing 7.3% of its corpus. These results show that **`Task Cascades`**—and particularly its **`Lite`** variant—recovers its investment quickly in realistic production settings.

E.2 Latency Measurements

Table 5 reports optimization and inference latencies for all methods, as well as estimated total inference latency for processing 1 million documents under 32-way concurrency. Measurements were taken on a 2024 MacBook Air (M4 chip) using 32 parallel API threads, with exponential back-off sleeps to handle rate limits. Note that API latencies can vary significantly across providers and time of day. **`Task Cascades`** optimization takes ~ 14× longer than **`2-Model Cascade`**, largely due to surrogate generation (o1-mini) and evaluating many candidate configurations. However, runtimes are not directly comparable across providers: as noted in Table 5, **`2-Model Cascade`**, **`Oracle Only`**, and **`Task Cascades (Lite)`** were measured on Azure OpenAI while **`Task Cascades`** used standard OpenAI endpoints; Azure can be ~2.5× slower [60]. This provider effect could explain why **`Task Cascades (Lite)`** shows a higher mean optimization runtime (1253s) despite its lower token cost.

For inference at scale at the 1M mark, our per-document measurements suggest that **`Task Cascades`** and **`Task Cascades (Lite)`** achieve within the same order of magnitude of latency (hours) to **`2-Model Cascade`**. However, these estimates should be interpreted cautiously: they do not account for API tail latencies or rate limiting, and the measurements were taken across different providers (Azure vs. standard OpenAI) with known latency differences. In batch processing scenarios—the primary use case for large-scale document processing—higher optimization latency is acceptable when it yields substantial cost savings, as demonstrated by the widespread adoption of batch APIs with extended completion times between 24 and 72 hours [20, 48].

F WORKLOAD PROMPT TEMPLATES

This section lists the exact prompt templates used for each classification workload in our experiments (Section 7). Workloads were selected as described in Section 7.

To ensure clear and consistent task specification, we used Claude 3.5 Sonnet to generate each instruction, explicitly prompting it to "make this an unambiguous classification task for an LLM." Each template includes a `{document_text}` placeholder, which was replaced by the input document at inference time.

The full prompt templates for all workloads are shown below.

AG NEWS

```
I will give you a news article. Here is the article: {document_text}

Assign the article to one of the following categories and return
      only the corresponding number:
- 0 = World: The article primarily discusses international news,
      global events, diplomacy, conflicts, or issues involving
      multiple countries or regions.
- 1 = Sports: The article is mainly about sporting events, teams,
      athletes, competitions, results, or sports-related news.
- 2 = Business: The article focuses on economic matters, companies,
      markets, finance, industry trends, or business-related
      developments.
- 3 = Sci/Tech: The article covers topics in science, technology,
      research, discoveries, innovations, or advancements in
      scientific or technological fields.

You must respond with ONLY the number (0, 1, 2, or 3) that best
      matches the main topic of the article.
```

Listing 1: AGNEWS classification prompt

Enron

```
I will give you an email. Here is the email: {document_text}

Your task is to determine if this email was sent from a senior
      executive or other high-ranking person at Enron.

- Return True if the email was sent from a senior executive (CEO,
      President, VP, Director, etc.) or other high-ranking person.
- Return False if the email was sent from a lower-level employee or
      non-executive.

Important notes: Look for job titles, positions, and signatures that
       indicate seniority. Consider both formal titles and contextual
       clues about the sender's role and authority level.

You must respond with ONLY True or False.
```

Listing 2: ENRON sender classification prompt

Court Opinion

```
I will give you a Supreme Court opinion. Here is the opinion: {
      document_text}

Your task is to determine whether this court opinion reverses a
      lower court's ruling.

Carefully read the opinion and consider the following:

- Return True if the Supreme Court (or the relevant higher court)
      reverses the decision of a lower court.
- Return False if the Supreme Court upholds (affirms) the lower
      court's ruling, or if the opinion does not address a lower
      court's decision.

You must respond with ONLY True or False.
```

Listing 3: COURT opinion reversal prompt

PubMed

```
I will give you a full biomedical research article from PubMed. Here
       is the article: {document_text}

Your task is to determine the type of biomedical study described in
      the full article.

Carefully read the article and determine which of the following
      study types best describes the research. Consider the study's
      methodology, data sources, and overall approach. Choose the
      single most appropriate type from the list below and return
      only the corresponding number:
- 0 = Randomized Controlled Trial (RCT): Participants are randomly
      assigned to groups to compare outcomes.
- 1 = Observational Study: Researchers observe existing groups
      without assigning interventions (includes cohort, case-control,
       cross-sectional).
```

```
- 2 = Meta-analysis or Systematic Review: Combines and analyzes
      results from multiple prior studies using systematic methods.
- 3 = Bench / Wet-lab Experimental Study: Laboratory-based
      experiments (e.g., cell culture, animal models, in vitro assays
      ).
- 4 = Computational / Bioinformatics Study: Uses computational
      models, simulations, or large-scale data analysis (e.g.,
      genomics, proteomics).
- 5 = Narrative Review (non-systematic): Describes a topic broadly
      without a structured or systematic review process.

You must respond with ONLY the number (0-5) that best matches the
      article's main study type.
```

Listing 4: PUBMED study type prompt

Game Reviews

```
I will give you a review for a game. Here is the review: {
      document_text}

Your task is to carefully read the following review and decide
      whether it mentions any other games in a more positive way than
       the game being reviewed.

Consider whether the reviewer compares the current game to another
      game and expresses a preference for the other game, either
      directly or indirectly. Look for statements that praise another
       game or suggest that the other game is better in some respect.

- Return True if the review references another game and describes it
       more favorably than the game being reviewed.
- Return False if the review does not mention other games, or if it
      does not express a preference for another game over the current
       one.

You must respond with ONLY True or False.
```

Listing 5: GAME review comparison prompt

Legal Docs

```
I will give you a legal document. Here is the document: {
      document_text}

Your task is to determine if this document contains any type of
      covenant not to sue or agreement not to challenge intellectual
      property rights. This includes both direct promises and
      indirect restrictions.

- True if it contains ANY of these:
  - Agreement not to contest/challenge IP validity or ownership
  - Promise not to question/attack/impugn IP rights
  - Agreement not to take actions inconsistent with IP ownership
  - Covenant not to bring claims/suits related to IP
  - More generally, any provision that could be interpreted as a
        restriction on future IP challenges
- False if it contains none of the above

You must respond with ONLY True or False.
```

Listing 6: LEGAL IP clause prompt

Wiki Talk

```
I will give you a Wikipedia Talk page discussion. Here is the
      discussion: {document_text}

Your task is to carefully read the following discussion and
      determine the outcome regarding the edits in question.

Consider whether the discussion led to a reversion (a rollback of
      previous edits) or resulted in a stable change to the content.

- Return True if the discussion resulted in reverting or rolling
      back changes to a previous version.
- Return False if the discussion led to stable changes being kept,
      or if no changes were made as a result of the discussion.

Be sure to look for explicit mentions of reversion, rollback, or
      restoration of prior content, as well as consensus to keep new
      changes.

You must respond with ONLY True or False.
```

Listing 7: WIKI_TALK edit revert prompt

FEVER

```
I will give you a claim and a list of documents that may or may not
      explicitly support the claim. Here is the claim and documents:
      {document_text}

Your task is to assess whether the provided claim is supported by
      the accompanying documents.

- Return True if at least one of the documents clearly supports the
      claim.
- Return False if none of the documents support the claim, or if the
       evidence is unclear or insufficient to determine support.

You must respond with ONLY True or False.
```

Listing 8: FEVER claim verification prompt