Dynamic Intent-Driven API Pruning for Tool Learning in Small Language Models

Tzu-Hao Wang[1] **Ting-Chun Yang**[1] **Sin-Syuan Wu**[1] **Ming-Der Yang**[2] **Yao-Chung Fan**[1]*
[1]Department of Computer Science and Engineering
[2]Smart Sustainable New Agriculture Research Center (SMARTer)
National Chung Hsing University, Taiwan
yfan@nchu.edu.tw

Abstract

Small Language Models (SLMs) struggle with tool calling, in part because a single API document often serves several usage intents and bundles the parameters of all of them into one schema, even though only a few are relevant to any given query. Providing the complete document forces the model to infer the intended usage scenario and its parameters at the same time, which lengthens the prompt and creates parameter ambiguity within a single API. We propose **Dynamic Intent-Driven API Pruning**. In an offline stage, an LLM decomposes each API document into *Intent Units*, each pairing an *Intent Scenario* with a *Key Parameter Subset*. At inference time, two-stage intent retrieval identifies the intents that match the query, and *Dynamic Flat Union* merges their parameter subsets into a query-specific pruned schema, keeping several intents rather than committing to a possibly incorrect top-1 choice. On 35 real-world APIs from ToolBench, our method improves argument correctness over Simulated Trial-and-Error while producing shorter prompts, with the gains concentrated on APIs that support multiple usage intents. Ablation studies confirm that both intent-driven pruning and Dynamic Flat Union reduce intra-API parameter ambiguity.

1 Introduction

Recent studies have improved tool calling primarily by enhancing *inter-tool* retrieval and organization (Liu et al., 2026; Wang et al., 2024). For example, Simulated Trial-and-Error (STE) (Wang et al., 2024) improves tool use through exploration trajectories and demonstration retrieval, while ToolScope (Liu et al., 2026) reduces semantic redundancy across tool collections via tool merging and hybrid retrieval. However, these approaches generally treat each API document as a monolithic unit. Even after retrieving the correct API, language models must still interpret the entire API document and identify the parameters relevant to the current user request. For APIs supporting multiple usage intents with numerous optional parameters, irrelevant information can interfere with parameter generation, leading to parameter hallucination (Qin et al., 2024), semantic entanglement, and attention dilution. These issues are particularly pronounced for Small Language Models (SLMs). Despite their importance, the organization of information *within* a single API (i.e., *intra-tool* organization) remains largely unexplored.

We observe that tool-calling failures are often caused not by incorrect API retrieval, but by difficulty identifying the relevant parameters after the correct API has already been selected. Many APIs support multiple usage scenarios with distinct parameter requirements, yet include all parameters within a single schema. As a result, the model must infer both the intended usage scenario and the corresponding parameters simultaneously, increasing parameter ambiguity and degrading tool-calling accuracy.

To address this issue, we propose **Dynamic Intent-Driven API Pruning**, a framework that extends tool organization from the inter-tool level to the intra-tool level. In the offline stage, a Large Language Model (LLM) decomposes each API document into multiple *Intent Units*, each consisting of an *Intent Scenario* and a corresponding *Key Parameter Subset*. During inference, the system performs two-stage intent retrieval and dynamically constructs a pruned API schema through *Dynamic Flat Union*. As a result, the language model receives only the parameters relevant to the current task, substantially reducing information redundancy and parameter ambiguity.

We evaluate our method on 35 real-world APIs from the ToolBench collection (Qin et al., 2024) and compare it with Simulated Trial-and-Error

* Corresponding author.

Proceedings of the 19th International Natural Language Generation Conference: Long Papers, pages 572–585
October 17-21, 2026 ©2026 Association for Computational Linguistics

(STE). Experimental results show that our framework improves argument generation quality while reducing prompt length without sacrificing overall tool-calling performance. The improvements are particularly significant for APIs with multiple usage intents. Ablation studies further demonstrate the effectiveness of both intent-driven API pruning and Dynamic Flat Union.

Our main contributions are summarized as follows:

- We propose **Dynamic Intent-Driven API Pruning**, which dynamically constructs a query-specific pruned API schema through offline intent decomposition, two-stage intent retrieval, and Dynamic Flat Union, reducing information redundancy in API documents and prompt complexity.
- We conduct comprehensive experiments on a real-world tool-calling benchmark, including comparisons with STE, analyses on complex APIs, and ablation studies. Results show that our framework improves argument generation for complex APIs while reducing prompt costs without compromising tool selection performance.

2 Related Work

2.1 Tool-Augmented Language Models

Tool-augmented language models extend the capabilities of LLMs by enabling them to interact with external resources such as search engines, databases, and third-party APIs during inference, allowing models to access up-to-date information and perform actions beyond their parametric knowledge (Mialon et al., 2023; Schick et al., 2023; Wang et al., 2024).

Existing approaches typically learn tool selection and argument generation through in-context learning (ICL) (Brown et al., 2020) or fine-tuning (Wang et al., 2024). While these methods have demonstrated promising performance, they generally rely on lengthy tool descriptions or large collections of demonstrations, causing prompt length to grow rapidly as the number of available tools increases. This limits scalability, increases inference cost, and can negatively affect the performance of small language models.

2.2 Tool Learning via Trial and Error

Simulated Trial-and-Error (STE) (Wang et al., 2024) guides language models to generate tool-use scenarios, execute tools, and reflect on the execution results, thereby collecting high-quality interaction trajectories to improve the model's ability to capture implicit constraints and generate correct arguments. However, STE-based inference pipelines still rely on complete API documents or retrieved demonstration trajectories, leaving the issues of lengthy prompts and inefficient knowledge reuse largely unresolved.

2.3 Hypothetical Knowledge for Retrieval

Recent studies have explored using Large Language Models (LLMs) to generate hypothetical knowledge to bridge the semantic gap between queries and documents. For instance, HyDE (Gao et al., 2023) generates a hypothetical document from the user query and performs dense retrieval (Karpukhin et al., 2020) using its embedding, significantly improving zero-shot retrieval performance.

Inspired by HyDE, our work generates an abstract *Hypothetical Scenario* instead of a complete document and semantically matches it against offline-constructed *Intent Scenarios*. Unlike HyDE, which targets document retrieval, our approach applies the same idea to API intent retrieval, improving intent identification and subsequent parameter pruning.

2.4 Tool Documentation Refinement

Recent work has explored improving tool utilization by refining tool documentation. EASY-TOOL (Yuan et al., 2025) identifies redundancy, inconsistency, and incompleteness in existing tool documentation and uses an LLM to transform raw documentation into concise tool descriptions and functionality guidelines, including parameter information and usage examples. This refined documentation improves tool retrieval, selection, and execution by providing LLMs with clearer and more structured tool instructions. However, the resulting tool instructions are constructed offline and remain static across different user queries. In contrast, our method focuses on query-dependent API organization: we decompose APIs into intent-specific units with corresponding parameter subsets during the offline stage, and dynamically retrieve relevant intents and prune irrelevant parameters according to each user query during inference. Thus, rather than constructing a single compact description for each API, our approach dynamically adapts the API schema to the current query, reduc-

ing parameter-level ambiguity for small language models.

2.5 Tool Organization for Tool Learning

Recent studies have explored tool organization to reduce the contextual burden of tool use for Large Language Models (LLMs). ToolScope (Liu et al., 2026) merges functionally similar tools and combines hybrid retrieval with reranking to reduce semantic redundancy and improve tool retrieval efficiency.

However, ToolScope focuses on *inter-tool* organization. After retrieving the correct tool, language models still need to interpret the complete API document, leaving parameter ambiguity caused by multiple usage scenarios within a single API unresolved.

In contrast, our work focuses on *intra-tool* organization. We decompose each API into intent-specific *Intent Units* and dynamically select the corresponding parameter subset during inference, reducing parameter-level ambiguity and improving argument generation.

3 Method

3.1 Problem Definition and Motivation

Our framework involves two language models of different scales, each serving a distinct role. A Large Language Model (LLM) acts as the *Intent Curator*, which analyzes API documents and defines intent units during the offline stage. A Small Language Model (SLM) serves as the *Inference Agent*, which receives the pruned API documents and performs the final tool call during online inference.

Given a collection of API documents $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, each API document d_i consists of a functional description and a set of parameter definitions. These parameters are divided into *Required Parameters*, which must be provided in every API invocation, and *Optional Parameters*, whose necessity depends on the specific usage scenario. The objective of tool calling is to guide the model, given a user query q, to select the target API d^* and generate the correct argument set a^*.

However, as discussed in Section 1, when a single API document simultaneously supports multiple usage scenarios and contains a large number of optional parameters, the resulting document complexity imposes a substantial input burden on the SLM, often leading to parameter hallucination and semantic entanglement. Based on this observation, we hypothesize that if the multiple usage intents within d_i can be decomposed offline and irrelevant parameters are dynamically pruned according to the current query q during inference, the reasoning burden of the SLM can be significantly reduced, thereby improving tool-calling accuracy.

To this end, we propose the *Dynamic Intent-Driven API Pruning* framework, illustrated in Figure 1. The framework consists of two major stages. In the offline stage, the LLM decomposes each monolithic API document into an intent repository with explicit parameter correspondences. During the online stage, the SLM retrieves the most relevant intent through semantic retrieval, dynamically removes irrelevant optional parameters, constructs a compact API schema, and injects it into the prompt to facilitate accurate tool calling.

3.2 Static Intent Definition

3.2.1 Intent Decomposition Criteria

For each API document d_i, a Large Language Model serving as the *Intent Curator* is prompted with the original API specification alongside a set of predefined heuristic instructions. The objective is to eliminate parameter-level ambiguity rather than exhaustively enumerate all possible scenarios. Specifically, the prompt enforces three core decomposition rules:

1. **Scenario-Based Decomposition:** APIs are partitioned by distinct user goals rather than individual filters. Decomposition occurs only when multiple scenarios share the same API specification but require substantially different key parameter subsets.

2. **Complexity Threshold:** To prevent over-fragmentation, APIs with a minimal parameter footprint default to a single intent ($K_i = 1$) and remain unified.

3. **Universal Parameter Preservation:** Core operational parameters (e.g., pagination or sorting) are strictly retained across all generated intents to maintain baseline functionality.

A complete decomposition example is provided in Appendix A, Figure 2.

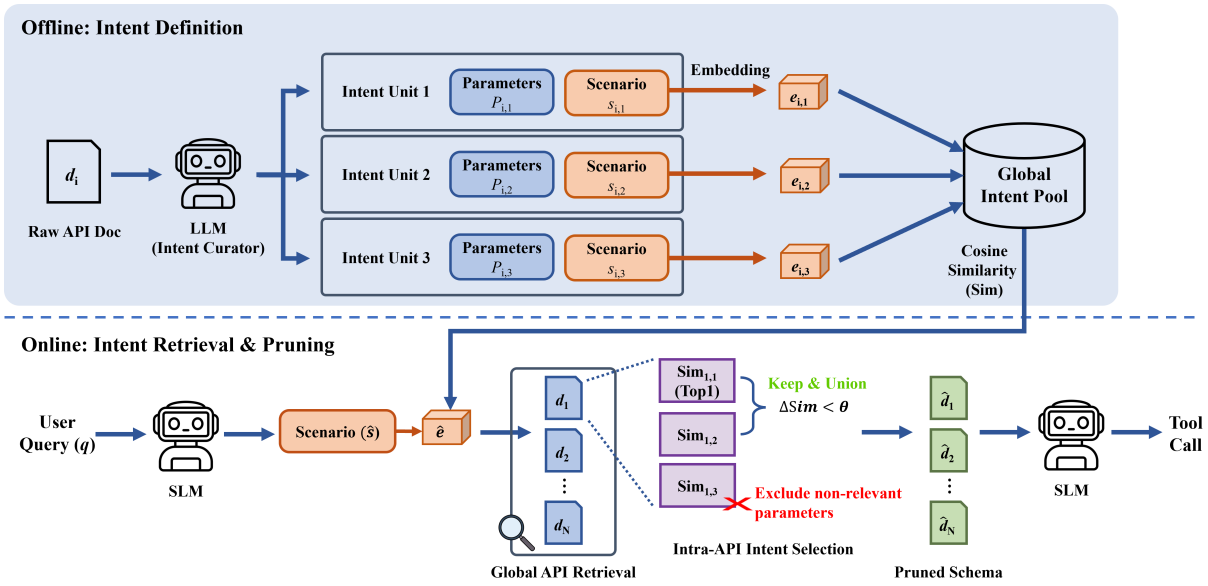


Figure 1: Overview of the proposed Dynamic Intent-Driven API Pruning framework. In the offline stage, an LLM acts as an Intent Curator to decompose raw API documents (d_i) into distinct intent units, each consisting of a parameter subset ($P_{i,k}$) and an intent scenario ($s_{i,k}$). Only the scenarios are embedded to serve as retrieval keys in the Global Intent Pool. In the online stage, an SLM generates a hypothetical scenario ($\hat{s}$) from the user query (q). The system performs global API retrieval and intra-API intent selection based on cosine similarity. Every intent whose similarity difference from the primary intent stays below the threshold (θ) is retained for the dynamic parameter union, while irrelevant parameters are excluded. The final pruned schemas are then injected into the SLM for accurate tool calling.

3.2.2 Intent Definition and Output

For each API document d_i, the Large Language Model (LLM) generates K_i intents, which together constitute the intent set of d_i:

$$\mathcal{I}_i = \{I_{i,k}\}_{k=1}^{K_i}, \quad I_{i,k} = (s_{i,k}, P_{i,k}).$$

where:

- $s_{i,k}$ denotes the k-th intent of d_i, represented as a natural language description from the user's perspective, which we define as the *Intent Scenario*. The scenario describes the user's goal under a particular usage context without referring to specific parameter names, thereby serving as the retrieval key for subsequent intent retrieval.
- $P_{i,k}$ denotes the *Key Parameter Subset* associated with the k-th intent of d_i. During the offline intent definition stage, the API's *Required Parameters* are statically injected into every parameter subset, together with the intent-specific key parameters.

The intent sets of all API documents collectively form the *Global Intent Pool* $\mathcal{R}$:

$$\mathcal{R} = \{I_{i,k} \mid 1 \le i \le n,\ 1 \le k \le K_i\}.$$

During the offline stage, the embedding vector of every Intent Scenario in $\mathcal{R}$ is precomputed:

$$\mathbf{e}_{i,k} = \mathrm{Embed}(s_{i,k}), \quad \forall I_{i,k} \in \mathcal{R}.$$

During inference, these precomputed embeddings can be directly retrieved without re-analyzing the corresponding API document d_i.

3.3 Hypothetical Scenario Generation and Intent Retrieval

3.3.1 Hypothetical Scenario Generation

User queries q are typically expressed as short, colloquial utterances, whereas intent descriptions $s_{i,k}$ are formulated as structured declarative sentences. This discrepancy creates an inherent semantic gap, making direct matching prone to retrieval errors. To address this issue, the online phase adopts the core idea of HyDE (Gao et al., 2023) by prompting the SLM to transform the user query q into an abstract *Hypothetical Scenario* $\hat{s}$:

$$\hat{s} = \mathrm{SLM}(q).$$

The generated $\hat{s}$ reformulates the original query from the perspective of the user's underlying intent and objective, thereby bridging the distributional gap between q and $s_{i,k}$ in the embedding

space and improving the accuracy of subsequent intent retrieval.

3.3.2 Global API Retrieval

The embedding vector of the hypothetical scenario is first computed as $\mathbf{e}_{\hat{s}} = \mathrm{Embed}(\hat{s})$. The system then measures its similarity against the embedding vectors $\mathbf{e}_{i,k}$ of all intent scenarios $s_{i,k}$ in the Global Intent Pool $\mathcal{R}$. For each API document d_i, the highest similarity score among its intent scenarios is selected as the representative score of the API:

$$\mathrm{Score}(d_i) = \max_{1 \le k \le K_i} \mathrm{Sim}(\mathbf{e}_{\hat{s}}, \mathbf{e}_{i,k})$$

where the similarity is computed using cosine similarity:

$$\mathrm{Sim}(\mathbf{e}_{\hat{s}}, \mathbf{e}_{i,k}) = \frac{\mathbf{e}_{\hat{s}} \cdot \mathbf{e}_{i,k}}{\|\mathbf{e}_{\hat{s}}\| \, \|\mathbf{e}_{i,k}\|}$$

The APIs are ranked according to $\mathrm{Score}(d_i)$ in descending order, and the top-N APIs are selected to form the candidate set $\mathcal{D}_{\mathrm{cand}}$.

3.3.3 Intra-API Intent Selection

For each candidate API $d_i \in \mathcal{D}_{\mathrm{cand}}$, the system further performs intent retrieval within the API by computing the semantic similarity between the hypothetical scenario $\hat{s}$ and every intent $I_{i,k}$. The intent with the highest similarity score is selected as the *Primary Intent*, denoted by I_i^*:

$$I_i^* = \operatorname*{argmax}_{I_{i,k} \in \mathcal{I}_i} \mathrm{Sim}(\mathbf{e}_{\hat{s}}, \mathbf{e}_{i,k}).$$

3.3.4 Similarity Difference Threshold

Semantically similar intents are located close to each other in the embedding space. Therefore, selecting only the Top-1 intent may lead to incorrect intent selection due to minor retrieval errors, causing relevant key parameters to be mistakenly pruned. Conversely, retaining too many intents may reintroduce optional parameters that should have been removed, thereby weakening the effectiveness of intent decomposition.

To address this issue, for each candidate API $d_i \in \mathcal{D}_{\mathrm{cand}}$, we dynamically construct an Active Intent Set $\mathcal{I}_{\mathrm{active}}$ based on a Similarity Difference Threshold θ. We first define the similarity difference between the Primary Intent I_i^* and another intent $I_{i,k}$ as

$$\Delta\mathrm{Sim}_{i,k} = \mathrm{Sim}(\mathbf{e}_{\hat{s}}, \mathbf{e}_i^*) - \mathrm{Sim}(\mathbf{e}_{\hat{s}}, \mathbf{e}_{i,k}). \quad (1)$$

All intents satisfying

$$\Delta\mathrm{Sim}_{i,k} < \theta \quad (2)$$

are retained to form the Active Intent Set:

$$\mathcal{I}_{\mathrm{active}} = \{ I_{i,k} \in \mathcal{I}_i \mid \Delta\mathrm{Sim}_{i,k} < \theta \}. \quad (3)$$

Here, $\mathbf{e}_i^*$ denotes the embedding of the Primary Intent I_i^*. For readability we suppress the API index i in $\mathcal{I}_{\mathrm{active}}$ and in the parameter sets derived from it, since the construction is carried out independently for each candidate API. This fixed similarity-difference threshold ensures that secondary intents are retained only when they are semantically close enough to the Primary Intent to fall within a reasonable retrieval error margin.

3.4 Dynamic Parameter Pruning and Injection

3.4.1 Active Parameter Set

For all intents in $\mathcal{I}_{\mathrm{active}}$, the system takes the union of their corresponding Key Parameter Subsets to construct the Active Parameter Set, denoted as P_{active}:

$$P_{\mathrm{active}} = \bigcup_{I_{i,k} \in \mathcal{I}_{\mathrm{active}}} P_{i,k}.$$

Since the Required Parameters are statically injected into every $P_{i,k}$ during the offline stage, the union operation naturally preserves all required parameters while merging intent-specific key parameters.

3.4.2 API Schema Pruning and Dynamic Flat Union

For the original API document d_i associated with $\mathcal{I}_{\mathrm{active}}$, the system replaces its parameter fields with the Active Parameter Set P_{active} to construct the pruned API Schema, denoted as $\hat{d}_i$. This operation preserves the original API name while removing all irrelevant parameters.

Meanwhile, the functional description of $\hat{d}_i$ is dynamically determined according to $|\mathcal{I}_{\mathrm{active}}|$:

- When $|\mathcal{I}_{\mathrm{active}}| = 1$, $\hat{d}_i$ directly adopts the intent-specific scenario description s_i^*. This represents the ideal case, where the SLM only needs to reason over a single task objective and the smallest possible parameter set, thereby minimizing input complexity.

- When $|\mathcal{I}_{\text{active}}| \geq 2$, the user query q is considered semantically ambiguous, lying near the boundary of multiple intents. In this case, $\hat{d}_i$ falls back to the original functional description of d_i, allowing the SLM to resolve the ambiguity based on the original API description together with the filtered parameter set P_{active}.

We refer to this strategy of maintaining a single schema while dynamically replacing only its parameter set and functional description as **Dynamic Flat Union**. This design preserves a flat API structure without introducing additional structural complexity, allowing the model to focus on generating values for the most relevant parameters.

Finally, the resulting pruned API Schema $\hat{d}_i$ is injected into the SLM prompt in place of the original API document d_i. Taking the mortgage API as an example (see Appendix B, Figure 3), when the `mortgage_from_home_value` intent is activated, the irrelevant parameter `loan_amount` is removed from the schema. Consequently, the model naturally maps the provided value to the correct `home_value` field, effectively eliminating semantic entanglement at the parameter level.

4 Experiments

4.1 Experimental Setup

We adopt Llama 3.1-8B (Grattafiori et al., 2024) as the online inference model and GPT-OSS-120B (Agarwal et al., 2025) as the offline *Intent Curator* for Intent Definition. GPT-OSS-120B is also used as an auxiliary judge for API calls requiring semantic evaluation.

Following Section 3, the offline stage constructs the intent sets ($\mathcal{I}_i$), the Global Intent Pool ($\mathcal{R}$), and precomputes embeddings for all Intent Scenarios. The temperature of GPT-OSS-120B is fixed at 0 during intent generation.

During inference, we follow the two-stage retrieval framework described in Section 3. The first stage retrieves the Top-N ($N = 15$) candidate APIs, while the second stage constructs the Active Intent Set ($\mathcal{I}_{\text{active}}$) using the Similarity Difference Threshold (θ). We set $\theta = 0.04$, retaining 92.67% of the ground-truth intents while preserving only 1.24 intents on average, providing a good balance between Intent Recall and Schema Compactness (see Appendix C, Table 6). Unless otherwise specified, all experiments use this configuration.

4.2 Dataset

We construct our evaluation dataset by manually selecting 35 APIs from ToolBench (Qin et al., 2024), a large-scale tool-learning benchmark built on the RapidAPI ecosystem, based on API documentation completeness, interface consistency, response format stability, and functional diversity. Each API is manually verified through actual API calls to ensure reliable accessibility throughout the experiments.

Based on the number of intents (K_i) obtained during the Intent Definition process (Section 3), we categorize APIs into *Simple APIs* ($K_i = 1$) and *Complex APIs* ($K_i > 1$) to evaluate our method under different levels of API complexity. The selected APIs span 16 functional categories.

For evaluation, we construct a balanced test set by generating 15 test queries for each of the 35 APIs, resulting in 525 evaluation instances. The queries are generated by GPT-4 (OpenAI, 2023) based on the API schema and functional description while avoiding explicit parameter names to better resemble natural language requests.

All generated queries are manually reviewed to ensure that the corresponding reference answers, consisting of API parameter values, are semantically consistent with the user queries.

Detailed API statistics, including category distribution and parameter complexity, are provided in Appendix D, Table 7.

4.3 Evaluation Metrics

We evaluate tool-calling performance using four metrics. **Well-formed Rate (WF)** measures whether the model output can be parsed as a valid structured tool call with JSON arguments. Outputs that directly answer the user without invoking an API are excluded.

Given well-formed outputs, **API-Match Rate (AM)** measures whether the predicted API matches the ground-truth API.

Argument Correctness (AC) evaluates argument generation over all test instances. For APIs with deterministic outputs (e.g., `verify_email`), predicted arguments are compared with the ground-truth using case-insensitive exact matching. For APIs requiring semantic judgment, GPT-OSS-120B serves as an auxiliary judge to determine whether the predicted API call and its arguments are semantically equivalent to the ground truth based on the API documentation and refer-

Table 1: Comparison between STE and our method on all 525 test queries. Best value per column in bold. Our method generates better arguments (AC) with shorter prompts (PT), at the cost of a marginally lower well-formed rate.

Method	WF (%)	AM (%)	AC (%)	PT
STE	**100.00**	96.76	76.95	6,262
Ours	99.24	**97.31**	**78.86**	**5,848**

Table 2: Argument Correctness on APIs with different intent complexity. Simple APIs contain a single intent ($K_i = 1$), while Complex APIs contain multiple intents ($K_i > 1$). The gain over STE is concentrated entirely on Complex APIs.

Method	Simple APIs (AC)	Complex APIs (AC)	Overall (AC)
STE	81.60	65.33	76.95
Ours	80.80	74.00	78.86

ence answer.

Finally, **Average Prompt Tokens (PT)** denotes the average number of prompt tokens per inference, measuring prompt length and inference cost.

To mitigate positional and recency biases, we randomly shuffle the order of candidate APIs during evaluation.

4.4 Baseline

We compare our method against Simulated Trial-and-Error (STE) (Wang et al., 2024), using Well-formed Rate (WF), API-Match Rate (AM), Argument Correctness (AC), and Average Prompt Tokens (PT) as the evaluation metrics.

We reimplement STE following its In-Context Learning (ICL) setting. The complete implementation details and hyperparameter configurations are provided in Appendix E. Our method adopts `all-MiniLM-L6-v2` (Sentence-Transformers, 2021a) as the embedding model for Intent Retrieval. The motivation for this choice and the corresponding comparison results are presented in Appendix F, Table 8.

During inference, we reproduce the Oracle Retriever setting used in the original STE paper to ensure that the ground-truth API is included in the Top-15 candidate set, thereby reducing the impact of API retrieval errors on tool-calling performance. Our Global API Retrieval also achieves 100% Recall@15 on all test instances (as detailed in Section 4.6), ensuring that both methods are evaluated under the same candidate API set.

In contrast, our method follows the procedure described in Section 3 by performing Intra-API Intent Selection and Dynamic Parameter Pruning over the candidate APIs. The resulting pruned API schema ($\hat{d}_i$), constructed through Dynamic Flat Union, is then provided to the language model as input.

4.5 Main Results

Table 1 presents the overall comparison between our method and STE. Under the same inference model (Llama 3.1-8B) and Oracle Retriever setting, our method improves Argument Correctness (AC) from 76.95% to 78.86%, an absolute gain of 1.91 percentage points, although this overall difference is not statistically significant (McNemar's test, $p = 0.378$). Meanwhile, Average Prompt Tokens (PT) decrease from 6,262 to 5,848, demonstrating that our method shortens prompt length while maintaining overall tool-calling performance.

Our method also improves API-Match Rate (AM) from 96.76% to 97.31%, indicating that Dynamic Intent-Driven API Pruning does not compromise tool selection. Although Well-formed Rate (WF) decreases slightly from 100.00% to 99.24%, the reduction is minor compared with the gains in Argument Correctness and prompt efficiency.

We further analyze the results by API complexity (Table 2). On Simple APIs ($K_i = 1$), our method achieves performance comparable to STE (80.80% vs. 81.60%; $p = 0.798$), suggesting limited benefit when an API contains only a single usage intent. In contrast, on Complex APIs ($K_i > 1$), Argument Correctness improves from 65.33% to 74.00%, an absolute gain of 8.67 percentage points, which approaches statistical significance ($p = 0.066$). Consistent with this trend, as shown in Table 3, prompt length remains comparable on Simple APIs (5,701 vs. 5,575) but is reduced by 22.1% on Complex APIs (6,217 vs. 7,978). All pairs are reported as ours vs. STE. These results indicate that prompt reduction mainly comes from removing usage scenarios and parameters irrelevant to the current query, leading to larger gains on APIs with multiple usage intents.

Table 3: Average prompt length (Prompt Tokens, PT) on APIs with different intent complexity. Prompt savings likewise come from Complex APIs, where pruning removes the parameters of the non-selected intents.

Method	Simple APIs (PT)	Complex APIs (PT)	Overall (PT)
STE	5,575	7,978	6,262
Ours	5,701	6,217	5,848

Table 4: Retrieval performance of the two-stage HyDE-based dense retrieval with `all-MiniLM-L6-v2`. API-level retrieval is saturated, so the remaining errors originate at the intent level.

Metric	**Value**
Global API Recall@15	100.00%
Intra-API Intent Recall@1	84.00%

4.6 Ablation Study

To better understand the contribution of each component in our framework, we conduct an ablation study by progressively removing or replacing key modules. Since Dynamic Intent-Driven API Pruning and Dynamic Flat Union are only applicable to APIs with multiple usage intents, all experiments in this section are conducted on the 10 Complex APIs ($K_i > 1$) in our dataset. All configurations use the same inference model (Llama 3.1-8B) and the same candidate API set $\mathcal{D}_{\text{cand}}$ (HyDE Top-15). As the Global API Retrieval achieves 100% Recall@15 (Table 4), all methods are evaluated end-to-end under the same candidate API set. In addition, the order of candidate APIs in the prompt is randomly shuffled to mitigate positional bias.

We use **Full Schema** as the reference configuration, where the complete API schema is directly provided without intent decomposition or parameter pruning. **Oracle Intent** constructs the pruned API schema using the ground-truth intent, serving as the upper bound of intent-guided parameter pruning.

Top-1 Intent retains only the Primary Intent with the highest similarity score, without applying Dynamic Flat Union. In contrast, **Dynamic Flat Union** extends Top-1 Intent by introducing a Similarity Difference Threshold ($\theta = 0.04$), retaining additional intents whose similarity difference from the Primary Intent is smaller than θ, and constructing the corresponding pruned API schema.

As shown in Table 5, Oracle Intent improves Argument Correctness (AC) from 70.00% to 78.67% compared with Full Schema, while reducing the average prompt length from 6,595 to 5,884 prompt tokens. This improvement is statistically significant (McNemar's test, $p = 0.019$), indicating that, when the correct intent is always identified, intent-guided parameter pruning effectively reduces parameter interference and improves argument generation quality.

Table 5: Ablation study on the Complex API subset ($K_i > 1$, $n = 150$ queries). * denotes a statistically significant improvement over the Full Schema baseline ($p < 0.05$, McNemar's test). †Oracle Intent is given the ground-truth intent and therefore serves as an upper bound; it is excluded from the bold-face comparison. Among the deployable configurations, Dynamic Flat Union recovers most of the gap between Top-1 Intent and the oracle.

Configuration	**WF (%)**	**AM (%)**	**AC (%)**	**PT**
Full Schema	98.67	**99.32**	70.00	6,595
Oracle Intent†	98.67	97.97	78.67*	5,884
Top-1 Intent	**99.33**	97.32	66.00	**5,889**
Dynamic Flat Union	98.67	**99.32**	**74.00**	6,217

Top-1 Intent, which removes Dynamic Flat Union, achieves an AC of 66.00%. Since the Intra-API Intent Retrieval achieves a Recall@1 of 84.00% (Table 4), the ground-truth intent is not ranked first for all queries. Consequently, retaining only the Top-1 Intent may incorrectly prune required parameters, leading to lower tool-calling accuracy.

In contrast, **Dynamic Flat Union** preserves candidate intents that are semantically similar to the Primary Intent, improving AC from 66.00% to 74.00% while increasing the average prompt length by only 328 tokens (5,889 $\rightarrow$ 6,217). This indicates a marginally significant improvement ($p = 0.052$) over Top-1 Intent, demonstrating that Dynamic Flat Union effectively compensates for intent retrieval errors with only a modest increase in prompt cost.

Furthermore, while Dynamic Flat Union yields a higher AC than the Full Schema baseline (74.00% vs. 70.00%), this difference does not reach statistical significance ($p = 0.424$) given the limited sample size of complex queries ($n = 150$). However, this observation aligns with our design objective: the dynamic pruning mechanism avoids degrading baseline performance even when intent retrieval is uncertain. We note a similar lack of statistical significance when comparing our method to the Oracle Intent upper bound ($p = 0.230$), indicating that a larger-scale evaluation is required

in future work to fully validate these performance boundaries.

5 Conclusion

In this paper, we propose **Dynamic Intent-Driven API Pruning** to address the parameter ambiguity encountered by Small Language Models (SLMs) during tool calling, where a single API document often contains multiple usage scenarios and numerous optional parameters. Unlike existing approaches that directly provide complete API documents during inference, our framework first decomposes each API document into multiple *Intent Units* with explicit usage scenarios during the offline stage. During inference, a two-stage intent retrieval process dynamically constructs a query-specific pruned API schema, allowing the model to process only the parameter subset relevant to the current task.

Experimental results show that our method improves argument generation accuracy while reducing prompt length and without compromising tool selection accuracy, at the cost of a marginally lower well-formed rate. The gains are concentrated on APIs that support multiple usage intents. Furthermore, the ablation study demonstrates the importance of both Dynamic Intent-Driven API Pruning and Dynamic Flat Union in reducing intra-API parameter ambiguity. These findings suggest that appropriate API document reorganization and dynamic parameter pruning can effectively reduce input complexity for Small Language Models during tool calling and improve their ability to handle complex APIs.

Limitations

This work focuses on reducing the complexity of individual API documents to improve tool calling for Small Language Models (SLMs). However, the offline Intent Definition stage still relies on a Large Language Model (LLM), and the resulting Intent Set depends on both the capability of the offline model and the prompt design. Although required parameters are statically injected into every Key Parameter Subset, different models or prompts may still produce different intent decompositions, affecting subsequent intent retrieval and API schema construction.

Furthermore, our evaluation is conducted on a tool collection built from publicly available and freely accessible APIs. The effectiveness of our approach on enterprise-level or commercial APIs remains to be investigated, and further evaluation on larger and more diverse tool ecosystems is needed to assess its generalizability. The evaluation set also remains modest in size, and several of the differences we report do not reach statistical significance; a larger-scale study is required to delineate these performance boundaries precisely.

As future work, our framework can be integrated with API organization methods such as ToolScope to jointly address both inter-tool and intra-tool organization. In addition, the intent annotations constructed in this work could be used to fine-tune a dedicated Intent Retriever, improving the accuracy and generalization of intent retrieval.

Acknowledgments

This work was supported in part by the National Science and Technology Council (NSTC), Taiwan, under Grants NSTC 115-2221-E-005-095-MY3 and NSTC 115-2218-E-005-002, and in part by the Delta Research Center, Delta Electronics, Inc. This work was also supported by the Smart Sustainable New Agriculture Research Center (SMARTer), funded by NSTC under Grant NSTC 114-2634-F-005-002.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

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

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and

Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 6769–6781.

Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Fahad Shah, and Dan Roth. 2026. ToolScope: Enhancing LLM agent tool use through tool merging and context-aware filtering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 34095–34119.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: A survey. *Transactions on Machine Learning Research*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Sentence-Transformers. 2021a. sentence-transformers/all-minilm-l6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Hugging Face model card.

Sentence-Transformers. 2021b. sentence-transformers/paraphrase-mpnet-base-v2. https://huggingface.co/sentence-transformers/paraphrase-mpnet-base-v2. Hugging Face model card.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2025. EasyTool: Enhancing LLM-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 951–972.

A API Intent Decomposition Example

This section provides a visual demonstration of our API intent decomposition process. Figure 2 illustrates how a monolithic API description is structured into multiple, purpose-specific intent units to mitigate semantic confusion.

B Parameter Generation Case Study

We conduct a qualitative evaluation to analyze the parameter generation behavior of small language models. Figure 3 presents a concrete error example under the monolithic baseline and demonstrates how our intent pruning framework resolves parameter confusion.

C Analysis of the Similarity Difference Threshold (θ)

This section provides detailed empirical results for selecting the Similarity Difference Threshold (θ) used in our two-stage intent retrieval framework. Table 6 illustrates the trade-off between Intent Recall and Schema Compactness across different threshold values.

D Dataset Statistics

The selected APIs span 16 functional categories, as summarized in Table 7. Based on the number of intents obtained during the Intent Definition stage, the dataset contains 25 *Simple APIs* ($K_i = 1$) and 10 *Complex APIs* ($K_i > 1$), providing a diverse benchmark for APIs with different levels of interface complexity.

In terms of parameter complexity, the number of parameters per API ranges from 0 to 17. Among the selected APIs, four require no parameters, six require only a single mandatory parameter, and the remaining APIs contain varying combinations of required and optional parameters. Across the 107 parameters in the dataset, `String` is the most common, accounting for 60 parameters (56.1%), followed by `Number` with 42 (39.3%), while `Boolean` (3, 2.8%) and `Enum` (2, 1.9%) appear less frequently.

E STE Reproduction Details

Following the original STE implementation, the offline exploration stage uses GPT-OSS-120B as the exploration model. Each API is explored for 10 exploration episodes, with up to 5 interaction

rounds per episode, using two short-term memory slots together with long-term memory.

During inference, Llama 3.1-8B is adopted as the tool-calling model. Following STE, `paraphrase-mpnet-base-v2` (Sentence-Transformers, 2021b) is used as the demonstration retriever to retrieve the Top-8 demonstrations for in-context learning.

F Embedding Model Selection

Although the original STE implementation adopts `paraphrase-mpnet-base-v2` (Sentence-Transformers, 2021b) for Demonstration Retrieval, our Intent Retrieval task emphasizes fine-grained semantic alignment within a single API. Therefore, we re-evaluate the performance of embedding models for this specific task. As shown in Table 8, although both models achieve a perfect Global API Recall@15 of 100%, `all-MiniLM-L6-v2` substantially outperforms `paraphrase-mpnet-base-v2` on Intra-API Intent Recall@1 (84.00% vs. 74.29%). Based on these results, we adopt `all-MiniLM-L6-v2` (Sentence-Transformers, 2021a) as the default embedding model in our framework.

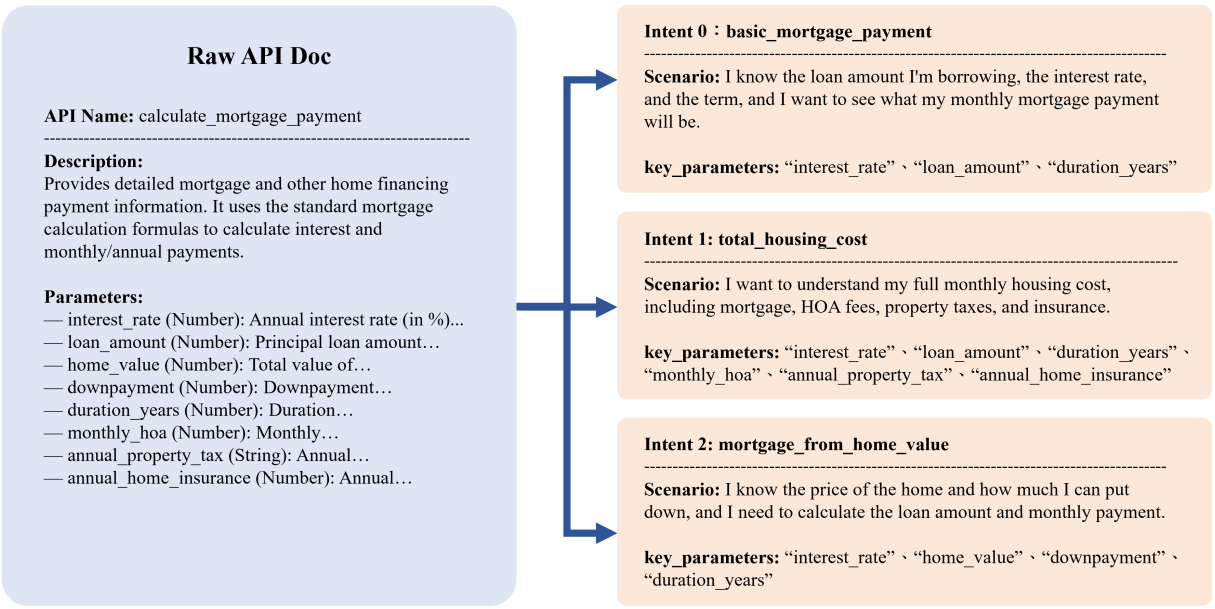


Figure 2: The original `calculate_mortgage_payment` API contains eight parameters covering multiple scenarios, which often overwhelms small language models. Our method decomposes it into three distinct intent units: (1) calculating payments from a loan amount, (2) deriving loan from home value, and (3) computing total ownership cost. By isolating intent-specific parameters, we replace the monolithic document with purpose-specific guides, reducing the prompt complexity for small models.

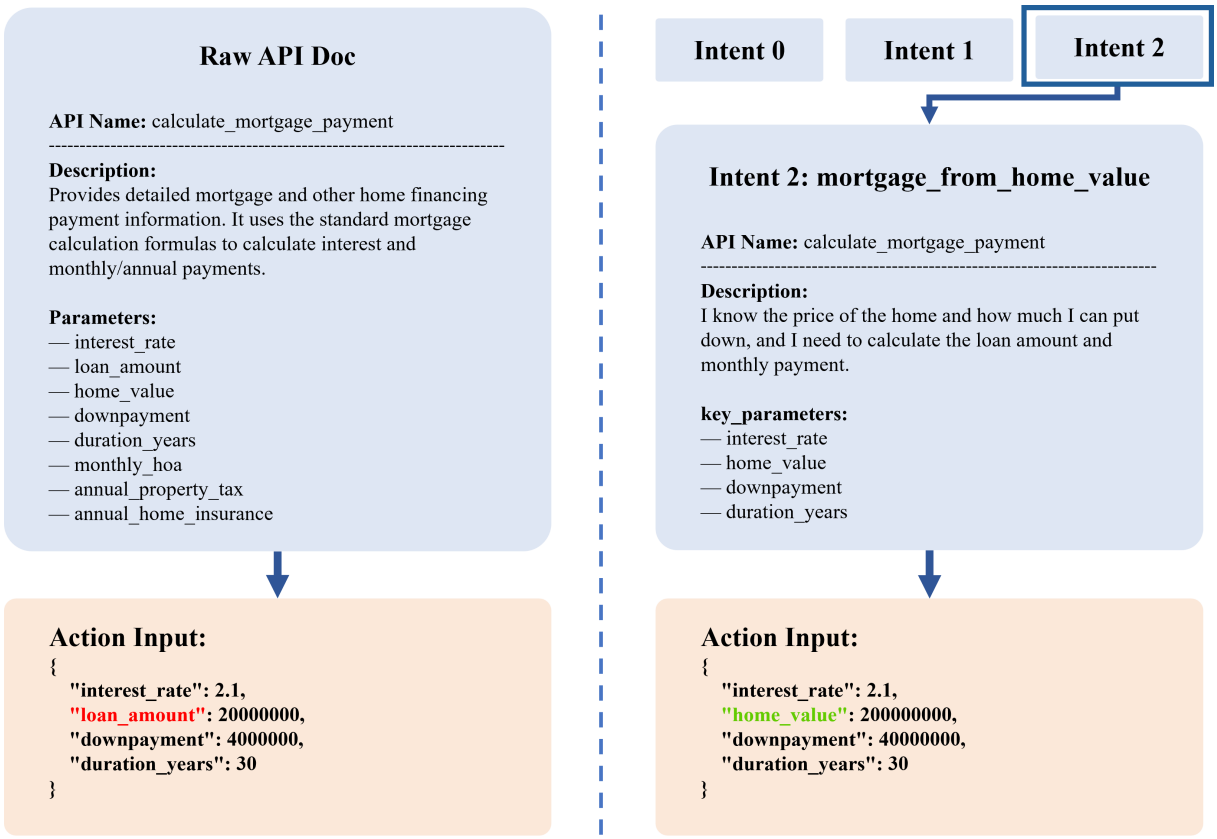


Figure 3: Case study comparing parameter generation on the query: "I plan to buy a house worth 20 million with a 4 million downpayment at 2.1% interest rate over 30 years. What is my monthly mortgage payment?" **Group A (Full Schema):** Presented with the monolithic API, the 8B model erroneously assigns the home value (20M) directly to `loan_amount`. This highlights a common failure where small models blindly map salient numbers to matching parameters without relational reasoning. **Group B (Intent-Pruned):** Our system retrieves the `mortgage_from_home_value` intent, exposing `home_value` while omitting `loan_amount`. Constrained to this relevant parameter set, the model correctly maps values to appropriate fields, effectively preventing semantic confusion.

Table 6: Effect of the Similarity Difference Threshold θ on intent retrieval. Recall denotes the percentage of test queries whose ground-truth intent is retained in the Active Intent Set. We adopt $\theta = 0.04$, which recovers most of the recall lost by Top-1 selection while still keeping close to a single intent per API.

θ	Recall (%)	Avg. Active Intents
0.00	84.00	1.00
0.01	86.67	1.05
0.02	89.33	1.11
0.03	90.00	1.17
0.04	92.67	1.24
0.05	93.33	1.30
0.06	94.00	1.35
0.08	98.00	1.48
0.10	98.67	1.61

Table 7: The 35 evaluation APIs and their functional categories. *Simple* denotes a single-intent API ($K_i = 1$) and *Complex* a multi-intent API ($K_i > 1$); the dataset contains 25 Simple and 10 Complex APIs across 16 categories.

Category	**Function**	**Type**
Business Software	`verify_email`	Simple
	`search_businesses`	Complex
Data	`get_animal_facts`	Simple
	`search_public_restrooms`	Simple
	`get_divisions_near_location`	Complex
Database	`get_dog_breeds_metadata`	Simple
	`search_exercises_by_name`	Simple
Entertainment	`get_media_news`	Simple
	`search_manga`	Simple
	`search_streaming_shows`	Simple
Finance	`get_financial_data`	Simple
	`convert_currency`	Simple
	`calculate_mortgage_payment`	Complex
	`get_trades_futures`	Complex
Food	`get_recipe`	Simple
	`search_cocktails`	Simple
	`get_restaurants_by_location`	Complex
Gaming	`get_filtered_game_giveaways`	Simple
	`get_lol_champion_stats`	Simple
Jobs	`get_salary_estimation`	Simple
Location	`get_timezone_info`	Simple
Science	`get_celestial_body_position`	Simple
	`get_planet_data`	Complex
Scholarly Search	`search_arxiv_papers`	Simple
Shopping	`search_amazon_products`	Simple
	`list_of_deals`	Complex
Sports	`get_handball_scheduled_matches`	Simple
Transportation	`get_airlines`	Simple
	`get_motorcycle_data`	Simple
	`calculate_route`	Complex
Travel	`get_airport_delay_statistics`	Simple
	`get_hotels_by_location`	Complex
	`get_flights_in_bounding_box`	Complex
Weather	`get_air_quality_data`	Simple
	`get_weather_forecast`	Simple

Table 8: Comparison of embedding models for intent retrieval. The two models are indistinguishable at the API level, but differ by almost ten points on intra-API intent selection, which is what parameter pruning depends on.

Embedding Model	**Global API Recall@15**	**Intra-API Intent Recall@1**
`paraphrase-mpnet-base-v2`	100.00	74.29
`all-MiniLM-L6-v2`	100.00	**84.00**