

ToolFuse: Separate-View Fusion of Documentation and User Intent for Tool Retrieval

Shao-Cheng Lu, Shui-Hsiang Hsu, Chen-Jui Yu, Yao-Chung Fan

National Chung Hsing University
Smart Sustainable New Agriculture Research Center (SMARTer)
Taichung, Taiwan
{g113056043, g113056055, rui0828}@smail.nchu.edu.tw
yfan@nchu.edu.tw

Abstract

Tool retrieval is a bottleneck for tool-augmented LLMs over large tool corpora, because a user query overlaps far less with a tool’s structured documentation than with an ordinary retrieval document. A common training-free remedy generates synthetic intent for each tool and concatenates it with the documentation before encoding, but this single pass mixes function-centric documentation and task-centric intent in one vector and leaves their contributions impossible to weight apart. We present **ToolFuse**, a tool retrieval framework with a tool-side and a query-side component. On the tool side, Complementarity-based Semantic Fusion (CSF) encodes a tool’s documentation and synthetic intent as two views and sets a per-tool fusion weight from their geometric complementarity. On ToolBench Per-Split, CSF matches the aggregate accuracy of fixed weights selected separately for the three pools without using query–tool labels. On the query side, ToolFuse splits a query that bundles several needs into focused sub-queries and aggregates the per-sub-query results by rank position. The full ToolFuse raises the average NDCG@5 of Qwen3-Embedding-0.6B from 64.72 to 75.49 and obtains the highest average NDCG@5 in all four evaluated encoder blocks. The same CSF plug-in also lifts the supervised ToolRetriever from 75.86 to 77.75 Per-Split, and the gain carries over to the larger Full-Corpus candidate pool.

1 Introduction

LLM systems increasingly call external tools to fetch live information and complete tasks (Qin et al., 2024a; Schick et al., 2023). When an inventory of thousands of APIs exceeds the context available for tool definitions, retrieval selects a small candidate set for planning and invocation (Qin et al., 2024b). This retrieval step is difficult because tools are described through semi-structured documentation of their functions, endpoints, and

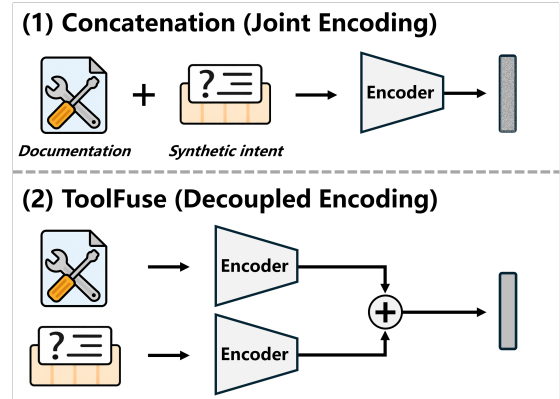


Figure 1: **Two ways to encode a tool.** (1) Text-level concatenation jointly encodes documentation and synthetic intent, leaving their relative contributions implicit. (2) ToolFuse encodes the two views separately and fuses their vectors with an explicit per-tool weight (CSF, §3.2).

parameters, whereas users express tasks in natural language. An analysis of TOOLRET illustrates this mismatch: query–documentation lexical overlap is lower than in several conventional IR benchmarks, and general-purpose retrievers transfer poorly to tool retrieval (Shi et al., 2025).

Two lines of work address this gap. Supervised methods learn query–tool correspondences from labeled pairs (Qin et al., 2024b; Wang et al., 2025; Fang et al., 2026). Updating a tool corpus requires refreshing its representations and may also require model updates when tool identities are encoded in model parameters. Generative augmentation draws on document expansion (Nogueira et al., 2019; Nogueira and Lin, 2019) to supply user-style phrasing. Re-Invoke (Chen et al., 2024b) applies this idea without tool-specific training: an LLM generates synthetic intents, each is concatenated with the documentation and encoded, and the resulting vectors are averaged.

Documentation and synthetic intent provide two views of the same tool. The documentation

view is function-centric and documentation-style, specifying functional boundaries, parameters, and schemas. The intent view is task-centric and user-style, describing situations in which a user would need the tool. Text-level concatenation joins these views before encoding, leaving their relative contributions to emerge implicitly from a single encoding pass. It therefore cannot explicitly discount an intent that largely restates already informative documentation. This motivates weighting the intent according to how much complementary information it adds for each tool.

We present **ToolFuse**, which replaces text-level concatenation with vector fusion of the two separately encoded views (Figure 1). Its **Complementarity-based Semantic Fusion (CSF)** sets a query-independent weight for each tool using geometric complementarity as a proxy for the information added by its intent view. Corpus-level standardization makes this weight relative to the other tools in the pool. The best fixed weight varies across ToolBench Per-Split pools. With query processing held fixed, CSF performs on par with fixed weights selected separately for each split, without using query-tool labels to set its weights (§6.2).

On the query side, a user request can bundle several functional needs, some of which may be underrepresented when encoded together. ToolFuse adopts Query Decomposition, drawing on prior work on topical decomposition and tool-related intent extraction (Bonchi et al., 2008; Chen et al., 2024b). It groups needs by tool type into focused sub-queries, retrieves against the fused index, and aggregates the lists by rank position so that candidates from different sub-queries can enter the final ranking. Decomposition is an established technique, and §5.3 characterizes when it helps.

Our main contributions are as follows.

- We replace text-level concatenation of documentation and intent with vector fusion of separately encoded views, and propose CSF to set each tool’s fusion weight from corpus-standardized geometric complementarity without per-corpus tuning.
- On ToolBench Per-Split, full ToolFuse, combining tool-side CSF with query-side decomposition, raises average NDCG@5 over documentation-only retrieval on all four encoders, by 10.8 to 15.8 points, including 64.72 to 75.49 on Qwen3-Embedding-0.6B.
- We add CSF as a representation-level plug-in

to the supervised ToolRetriever, raising Per-Split average NDCG@5 from 75.86 to 77.75, and the gain holds when the candidate pool expands to Full-Corpus.

2 Related Work

2.1 Learning to Match Queries and Tools

Training-based tool retrieval learns query-tool correspondence from labeled pairs or tool-use data. ToolRetriever fine-tunes a dense bi-encoder on query-documentation pairs (Qin et al., 2024b). ToolRerank combines learned relevance scoring with hierarchy-aware reranking (Zheng et al., 2024), while COLT learns tool relationships through graphs linking queries, tool sets, and tools (Qu et al., 2024). Generative retrieval instead trains an LLM to produce tool identifiers, using individual tokens in ToolGen (Wang et al., 2025) or hierarchical code sequences in ToolWeaver (Fang et al., 2026). Table 1 summarizes representative methods.

Tool2Vec uses the average embeddings of example queries associated with each tool and combines this representation with a trained ToolRefiner (Moon et al., 2024). ToolFuse instead generates task-centric intents from documentation and retains the documentation as an independent view. It requires no labeled example queries for tool representation construction and also applies to an already fine-tuned encoder (§5.2).

2.2 Generative Augmentation and View Fusion

Generative augmentation supplies alternative expressions for retrieval. Document expansion generates likely queries for documents (Nogueira et al., 2019; Nogueira and Lin, 2019), whereas Query2doc and HyDE generate pseudo-documents from a search request (Wang et al., 2023; Gao et al., 2023). Re-Invoke extends document expansion to tool retrieval by jointly encoding each generated intent with the documentation and averaging the vectors (Chen et al., 2024b). It is the closest tool-retrieval reference for our representation design.

Expansion is not uniformly beneficial across retrieval settings (Weller et al., 2024), motivating control over how generated text contributes to a representation. ToolFuse makes this contribution explicit by separately encoding and weighting documentation and intent. The controlled comparison in §6.1 evaluates this construction against Re-Invoke’s joint encoding while holding the gener-

Method	TF	LLM@q	Tool repr.	Query op.	Family
<i>Training-based retrieval</i>					
ToolRetriever (Qin et al., 2024b)	✗	0	doc only	–	dense
Tool2Vec + ToolRefiner (Moon et al., 2024)	✗	0	query avg.	–	dense + reranker
COLT (Qu et al., 2024)	✗	0	doc + graph	–	graph
ToolRerank (Zheng et al., 2024)	✗	0	doc only	–	reranker
ProTIP (Anantha et al., 2023)	✗	0	doc only	implicit decomp.	dense
ToolGen (Wang et al., 2025)	✗	–	token	–	generative
<i>Iterative retrieval</i>					
AnyTool (Du et al., 2024)	✓	iterative	–	–	agent
IterFeedback (Xu et al., 2024)	✗	iterative	doc only	rewrite	dense
<i>Generative augmentation</i>					
Re-Invoke (Chen et al., 2024b)	✓	single	concat	decomposition	dense
ToolFuse (ours)	✓	single	fusion	decomposition	dense

Table 1: **Representative methods grouped by primary retrieval intervention.** *TF*: no retrieval component is trained on the tool corpus. *LLM@q*: extra LLM calls for retrieval, excluding downstream planning; ToolGen integrates generation into the retriever. *Tool repr.*: documentation as tool-side text (*doc only*), documentation embeddings refined by graph learning (*doc + graph*), average example-query embedding (*query avg.*), jointly encoded documentation and intent (*concat*), separately encoded and fused views (*fusion*), or learned identifier (*token*). A dash (–) denotes not applicable.

ated intents and query processing fixed.

Prior retrieval models learn representations of multiple document fields (Zamani et al., 2018), fuse learned aspect embeddings (Kong et al., 2022), or retain multiple document views for query matching (Zhang et al., 2022). MFAR learns query-conditioned weights for field-level scores (Li et al., 2025). Closer to our setting, STAR separately encodes table content and generated queries without retrieval training; its dynamic fusion rule increases the generated-query weight with similarity between the views (Hsu et al., 2026). CSF instead uses the intent view’s residual beyond the documentation direction as a complementarity proxy, standardized against the tool corpus. Its distinction is this training-free, query-independent weighting rule, whose comparison with tuned fixed weights appears in §6.2.

2.3 Query-Side Retrieval Strategies

Iterative methods use retrieval feedback to rewrite queries (Xu et al., 2024) or reflection on failed solutions to guide hierarchical search (Du et al., 2024), potentially requiring repeated LLM calls. Topical query decomposition instead produces queries covering distinct topics (Bonchi et al., 2008). In tool retrieval, ProTIP learns implicit decomposition through progressive updates to the query embedding (Anantha et al., 2023), while Re-Invoke explicitly extracts tool-related intents and ranks tools by within-intent rank, then similarity (Chen et al.,

2024b).

ToolFuse adopts explicit Query Decomposition through one LLM call and uses the same rank-first, similarity-second aggregation principle as Re-Invoke. §3.3 details the procedure, and Appendix C.5 compares aggregation alternatives. We also evaluate hierarchical routing and cross-encoder reranking as alternative retrieval procedures (§4.2). These query-side choices complement CSF’s tool-side representation design.

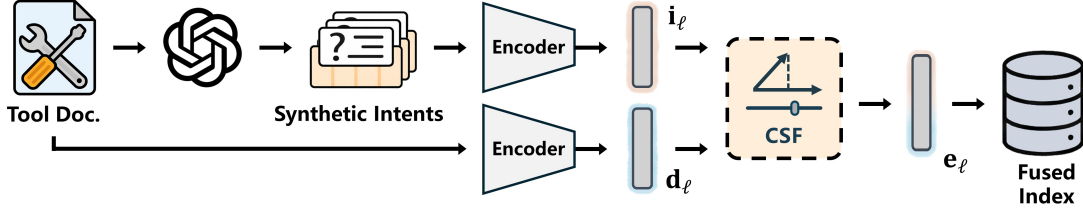
3 Method

ToolFuse runs in an offline and an online stage that share a single encoder, as shown in Figure 2. The offline tool-side stage applies CSF to combine each tool’s function-centric documentation view and task-centric intent view into one vector. The online query-side stage adopts Query Decomposition to split a multi-intent query into focused subqueries before retrieval. We detail the tool-side construction in §3.2 and the query-side procedure in §3.3.

3.1 Problem Formulation

Given a tool corpus $\mathcal{T} = \{d_\ell\}_{\ell=1}^N$ with tool index $\ell \in \{1, \dots, N\}$, the system receives a user query Q and must return a top- K ranking of the tools relevant to it. We work in a dense bi-encoder setting with one precomputed embedding per tool. A frozen dense encoder f maps the query and tools to vectors, and tools are ranked by their similarity

(1) Offline Stage: Tool Representation Construction



(2) Online Stage: Query Decomposition and Aggregation

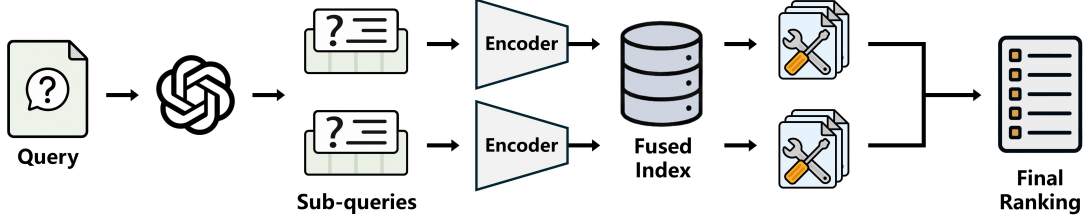


Figure 2: **The ToolFuse architecture.** Offline, a frozen dense encoder maps each tool’s documentation and its LLM-generated synthetic intents into a documentation view $\mathbf{d}_\ell$ and an intent view $\mathbf{i}_\ell$. CSF reads a per-tool fusion weight from how much the intent view extends beyond the documentation direction, standardized against the corpus, and combines the two views into one fused vector $\mathbf{e}_\ell$ per tool. Online, an LLM splits the query into sub-queries at a per-tool-type granularity; each retrieves over the fused index, and rank aggregation produces the final ranking.

to the encoded query.

3.2 Offline Stage: Tool Representation Construction

The offline stage precomputes one fused vector per tool. It first builds the two complementary views of the tool and then combines them into a single vector.

Documentation view and intent view. We treat the documentation and the synthetic intent as two semantic views of the same tool and encode each separately. The documentation view is the embedding $\mathbf{d}_\ell = f(d_\ell)$ of the tool documentation. It gives a function-centric representation of the tool’s functional boundaries, calling conditions, parameters, and schema. For the intent view, the LLM generates n synthetic intents $u_\ell^1, \dots, u_\ell^n$ for d_ℓ , each a user-style search query. The intent combination step joins them into a single text $c_\ell = \text{Concat}(u_\ell^1, \dots, u_\ell^n)$, encoded once as $\mathbf{i}_\ell = f(c_\ell)$, a task-centric representation of the tool’s typical usage contexts. The intent view is therefore a single vector of the same dimensionality as the documentation view, and separate encoding leaves both available for explicit fusion. Retrieval scores tools by cosine similarity, so both view vectors are L2-normalized, and $\mathbf{d}_\ell$ and $\mathbf{i}_\ell$ denote the normalized vectors below. Appendix C.1 reports the sweep over n and an alternative intent combination, and

Appendix A.1 gives the full prompt and examples.

Complementarity-based Semantic Fusion (CSF).

Tools differ in how complete their documentation is and in how much user-style information their synthetic intent adds. A single shared fusion weight assigns the same value to every tool and so ignores this tool-level diversity. CSF instead estimates a query-independent fusion weight for each tool from the geometry of its two views in embedding space alone.

We use the intent view’s deviation from the documentation direction as a geometric proxy for complementarity, decomposing $\mathbf{i}_\ell$ into a component aligned with $\mathbf{d}_\ell$ and an orthogonal residual, and score complementarity by the norm of that residual,

$$s_\ell = \|\mathbf{i}_\ell - (\mathbf{i}_\ell \cdot \mathbf{d}_\ell) \mathbf{d}_\ell\|, \quad (1)$$

which is the sine of the angle between the two views. §6.2 examines how the resulting fusion weight behaves in retrieval, and Appendix Table 9 reports an alternative complementarity score based on cosine distance.

Corpus-level standardization and vector fusion.

CSF interprets each s_ℓ relative to the candidate corpus rather than through an absolute threshold. It standardizes the scores with a corpus-level z-score $z_\ell = \frac{s_\ell - \mu}{\sigma}$ and maps the result through a temperature sigmoid into a fusion weight $\alpha_\ell =$

$\text{sigmoid}(\frac{z_\ell}{\tau})$, where μ and σ are the mean and standard deviation of $\{s_\ell\}_{\ell=1}^N$. A tool at the corpus mean maps to $\alpha_\ell \approx \frac{1}{2}$, and above-average complementarity earns a higher intent-side weight. The temperature τ controls the smoothness of the sigmoid. A larger τ pulls every α_ℓ toward the neutral value $\frac{1}{2}$, and a smaller one makes it more sensitive to z-score deviations. We set $\tau = 4$ globally, and Appendix C.3 reports the sensitivity sweep.

The fused representation of a tool is the weighted interpolation of its two views,

$$\mathbf{e}_\ell = (1 - \alpha_\ell) \mathbf{d}_\ell + \alpha_\ell \mathbf{i}_\ell, \quad (2)$$

so $\mathbf{e}_\ell$ directly replaces the per-tool vector of a documentation-only baseline. The fused index $\{\mathbf{e}_\ell\}_{\ell=1}^N$ is built once offline per corpus and is then shared by all queries.

3.3 Online Stage: Query Decomposition and Aggregation

Query Decomposition. A complex query can bundle several functional needs that map to different tool types. We use the LLM to split Q into m sub-queries $\tilde{q}_1, \dots, \tilde{q}_m$, each capturing one such type; the LLM determines m from the query. A single-need query produces one rewritten sub-query. The prompt groups needs handled by the same tool type, so checking hotel prices in several cities stays together while hotel search and EV-charger search split apart. Sub-queries remain in user-style phrasing and are encoded directly. Appendix A.3 gives the full prompt template and examples.

Per-sub-query retrieval. For each sub-query $\tilde{q}_k$ we compute $\tilde{\mathbf{q}}_k = f(\tilde{q}_k)$ and take the top- L results on the fused index by cosine similarity, written as the ranked list

$$R_k = ((t_k^{(1)}, r_k^{(1)}), \dots, (t_k^{(L)}, r_k^{(L)})),$$

where $t_k^{(j)}$ is the j -th ranked tool for sub-query $\tilde{q}_k$, $r_k^{(j)}$ is its cosine similarity, and L is the per-sub-query candidate cutoff.

Rank aggregation. Decomposition produces one ranked list per sub-query, $R_1, \dots, R_m$. Ranking all candidates by raw similarity could let a sub-query with generally higher scores dominate the final list. Following Re-Invoke’s rank-first, similarity-second principle (Chen et al., 2024b), we give priority to each candidate’s position in its own list. We first

consider every sub-query’s first-ranked candidate, then its second-ranked candidate, and so on, one round per rank. At each rank, we order candidates by cosine similarity to their respective sub-queries, skip tools already selected, and stop after K distinct tools. Similarity therefore determines the order only among candidates at the same rank and, when those candidates outnumber the remaining slots, which of them are retained. Appendix C.5 gives the pseudocode and compares this procedure with its version without similarity sorting, RRF, and CombSUM.

4 Experimental Setup

4.1 Datasets and Splits

Our primary evaluation uses **ToolBench** (Qin et al., 2024b), which provides three test splits: **I1** (single-tool instructions), **I2** (intra-category multi-tool), and **I3** (intra-collection multi-tool). These names describe how each split’s instructions were composed over the tool inventory, not the number of ground-truth tools per query. §5.3 tests transfer to **TOOLRET** (Shi et al., 2025), with the full table in Appendix C.7.

We evaluate under two candidate pools. Under *Per-Split*, our main setting, the pool $\mathcal{T}_i$ holds only the tools referenced by the labeled samples of split $i \in \{I1, I2, I3\}$, which supports controlled comparison with prior results. Under *Full-Corpus*, the pool $\mathcal{T}_{\text{full}}$ is the unfiltered RapidAPI crawl released with ToolBench (49,415 endpoints), which is larger than the curated API set the benchmark itself retains, so each test query is scored against tens of thousands of tools that no split references. Full-Corpus mainly probes the supervised ToolRetriever, fine-tuned on the ToolBench training set, against this enlarged pool (§5.2); Appendix C.6 reports the training-free methods there. Appendix Table 5 reports pool sizes and query counts, and we report the standard ToolBench metric **NDCG@K** ($K \in \{1, 3, 5\}$). Retrieval averages give equal weight to I1, I2, and I3; downstream averages are weighted by subset query counts.

4.2 Retriever Backbones and Comparison Setup

Our core comparison focuses on first-stage retrieval with one precomputed embedding per tool and spans four dense encoders: Qwen3-Embedding-0.6B (Zhang et al., 2025), BGE-M3 (Chen et al., 2024a), mE5-large-instruct (Wang et al., 2024), and

Method	I1			I2			I3			Average		
	@1	@3	@5	@1	@3	@5	@1	@3	@5	@1	@3	@5
BM25	29.46	34.11	36.76	24.13	29.54	32.31	32.00	32.77	38.33	28.53	32.14	35.80
Qwen3-Embedding-0.6B												
plain	63.83	65.47	74.21	49.37	53.57	58.20	52.00	55.83	61.76	55.07	58.29	64.72
Re-Invoke*	64.67	66.38	71.21	54.14	59.60	64.18	53.00	55.71	65.37	57.27	60.56	66.92
Hierarchical [†]	53.00	54.05	58.03	47.12	48.57	51.97	57.00	58.40	63.81	52.37	53.67	57.94
Reranker [†]	60.85	66.03	72.47	53.68	58.07	64.63	71.00	68.49	73.99	61.84	64.20	70.36
ToolFuse (ours)	67.33	69.25	76.23	61.15	62.20	71.47	69.00	67.87	78.76	65.83	66.44	75.49
BGE-M3												
plain	58.00	58.73	64.02	46.12	48.13	52.86	43.00	51.16	58.78	49.04	52.67	58.55
Re-Invoke*	59.00	63.47	68.04	58.90	62.48	67.11	54.00	56.34	62.30	57.30	60.76	65.82
Hierarchical [†]	27.67	27.54	29.91	25.56	24.68	26.80	27.00	34.58	38.58	26.74	28.93	31.76
Reranker [†]	60.42	65.87	72.10	53.69	58.78	64.15	72.00	67.95	74.38	62.04	64.20	70.21
ToolFuse (ours)	63.50	66.05	73.84	61.15	62.03	70.68	67.00	65.51	77.04	63.88	64.53	73.85
mE5-large-instruct												
plain	62.33	62.31	66.72	47.12	50.75	54.44	51.00	48.18	54.04	53.48	53.75	58.40
Re-Invoke*	60.08	61.03	65.94	56.64	59.16	63.98	56.00	52.40	60.69	57.57	57.53	63.54
Hierarchical [†]	39.67	38.54	41.41	39.10	40.71	44.21	45.00	45.94	50.29	41.26	41.73	45.30
Reranker [†]	61.71	66.51	72.65	52.59	57.67	63.75	69.00	65.64	72.29	61.10	63.27	69.56
ToolFuse (ours)	66.83	68.20	74.80	61.15	61.21	70.45	62.00	68.73	77.25	63.33	66.05	74.17
GTE-large												
plain	53.67	56.89	61.74	42.36	43.99	49.03	48.00	49.13	55.72	48.01	50.00	55.50
Re-Invoke*	47.00	50.26	55.30	39.10	45.64	51.96	54.00	54.22	58.76	46.70	50.04	55.34
Hierarchical [†]	29.67	30.76	33.35	34.34	34.18	37.43	36.00	40.17	43.99	33.34	35.04	38.26
Reranker [†]	60.45	66.01	71.74	52.08	56.84	62.29	72.00	69.40	74.15	61.51	64.08	69.39
ToolFuse (ours)	60.50	63.17	70.31	50.63	56.06	64.70	61.00	65.27	73.26	57.38	61.50	69.42

Table 2: **ToolBench tool retrieval across four dense encoders (Per-Split, NDCG@K, %)**. All methods are training-free, and the dense methods use the encoder of their block; §4.2 defines them. [†]Hierarchical and Reranker change the retrieval procedure rather than the tool representation. *Re-Invoke is our reproduction, not the authors’ reported numbers. Bold marks the best value per column within each encoder block; BM25 is not bolded.

GTE-large (Li et al., 2023; Zhang et al., 2024). Within each block, *plain*, **Re-Invoke** (Chen et al., 2024b), and **ToolFuse** share this operating point, so their differences come from tool representation and query processing alone. *plain* retrieves the original query against the documentation index, and we reproduce Re-Invoke with its original prompts and query-side recipe on each encoder. Qwen3-Embedding-0.6B (Qwen3 below) is the default for §6, and **BM25** (Robertson and Zaragoza, 2009) provides a sparse reference.

We add two training-free designs that keep the encoder but change the retrieval procedure. **Hierarchical** routes through ToolBench’s native category tree, taking the top five categories, then 20 tools from their names and descriptions, then the full tool records including parameters. **Reranker** re-scores the plain dense top 100 with the zero-shot cross-encoder Qwen3-Reranker-0.6B (Zhang et al., 2025), which adds about 100 cross-encoder passes per query on top of the dense retrieval stage. Neither updates any weight, so both stay training-free. Both also run under Full-Corpus, and the router ad-

ditionally runs end to end on Qwen3. Appendix B lists model identifiers and dimensions.

As a supervised reference we compare against **ToolRetriever** (Qin et al., 2024b), a dense bi-encoder fine-tuned on the ToolBench training-set query-tool pairs. Because it stores one pre-computed embedding per tool, we can add CSF as a tool-side plug-in (denoted + **CSF**) to test whether CSF and supervision are complementary (§5.2). The plug-in encodes both views with ToolRetriever’s own encoder and replaces its documentation-only index with the fused vectors, leaving the rest of the procedure untouched.

4.3 Implementation Details

For the ToolBench experiments, the default encoder is Qwen3-Embedding-0.6B (1024-dimensional), and both offline synthetic intent generation and on-line Query Decomposition use GPT-4o-mini. We generate $n = 10$ synthetic intents per tool and set the fusion temperature $\tau = 4$, both fixed across all splits and candidate pools. Appendices C.1 and C.3 report the two sweeps. At query time each

sub-query retrieves its top- L candidates from the fused index with $L = 50$, and rank aggregation produces the final top- K ranking with $K = 5$ (§3.3). Following each encoder’s official recommendation, Qwen3-Embedding and mE5-large-instruct prepend their suggested task instruction prefix on the query side (Appendix A.5), while BGE-M3 and GTE-large use none; the prefix is identical across every row of a block. BM25, the two alternative designs, and ToolRetriever follow their standard settings, and Appendix Table 6 lists the core hyperparameters.

5 Main Results

5.1 ToolFuse under the Training-Free Setting

Under Per-Split (Table 2), ToolFuse reaches the highest Average NDCG@5 in every encoder block: 75.49 on Qwen3-Embedding-0.6B, 73.85 on BGE-M3, 74.17 on mE5-large-instruct, and 69.42 on GTE-large. Against plain retrieval and Re-Invoke, it leads at every split and cutoff but one, BGE-M3 I2 at NDCG@3. Its Average NDCG@5 gain runs from +10.8 to +15.8 points over plain retrieval and from +8.0 to +14.1 over Re-Invoke.

Re-Invoke raises I2 and I3 NDCG@5 on all four encoders but improves I1 only on BGE-M3, so on GTE-large the I1 drop cancels its other gains and leaves the average nearly unchanged (55.34 vs. 55.50 for plain retrieval). ToolFuse instead improves all three splits on all four encoders, including I1 by 2.0–9.8 points. Re-Invoke also splits queries into per-need intents, so its margin to ToolFuse mainly reflects tool representation; §6.1 isolates that part with the query side held fixed.

Among the alternative procedures, routing through ToolBench’s own categories is the weaker design on every encoder (57.94 down to 31.76 Average NDCG@5) and stays below plain flat retrieval even though its last stage reads the full tool records, consistent with a tool dropped at the category level being unrecoverable. The reranker reaches 69.39 to 70.36 and takes most I3 cells, although ToolFuse leads I3 NDCG@5 on three encoders. ToolFuse also keeps the higher average on all four encoders, by 5.13 points on Qwen3 and level on GTE-large (69.42 vs. 69.39). It does so with one similarity search per sub-query plus one LLM decomposition call, instead of scoring 100 query–candidate pairs after dense retrieval.

On Qwen3, Query Decomposition contributes +8.78 over documentation-only retrieval (64.72 to

Query	Tool repr.	I1	I2	I3	Avg.
<i>Per-Split</i>					
Direct	Documentation	83.86	78.54	65.19	75.86
Decomp.	Documentation	78.63	75.41	69.86	74.63
Direct	CSF	84.19	77.94	71.12	77.75
Decomp.	CSF	80.21	76.29	72.08	76.19
<i>Full-Corpus</i>					
Direct	Documentation	76.88	70.03	57.75	68.22
Decomp.	Documentation	67.41	67.65	50.79	61.95
Direct	CSF	77.82	70.29	62.78	70.30
Decomp.	CSF	71.29	68.56	50.37	63.41

Table 3: **Supervised ToolRetriever under both candidate pools (NDCG@5, %; best per column within each pool in bold).** We cross query processing with tool representation.

73.50) and the CSF-fused index a further +1.99 (73.50 to 75.49), or +1.73 without decomposition (64.72 to 66.45). The query side thus provides the larger share on Qwen3, which is also where the tool-side contribution is smallest: the other three encoders gain +4.9 to +6.5 from it (Appendix C.8).

5.2 CSF as a Plug-in on a Supervised Retriever

We next test whether tool-side fusion still helps after supervised fine-tuning, and whether that gain survives the Full-Corpus pool. Table 3 crosses two query-processing choices with two tool representations under both pools, so CSF is compared with documentation-only retrieval at fixed query processing. Under Per-Split, CSF raises average NDCG@5 from 75.86 to 77.75 with direct queries and from 74.63 to 76.19 with decomposition. Direct queries with CSF give the highest average; the largest split-level gain is on I3 (65.19 to 71.12), despite a decrease on I2. Under Full-Corpus every configuration scores lower, but direct-query CSF again improves the average (68.22 to 70.30) and each split, with I3 gaining the most (57.75 to 62.78). The tool-side improvement therefore persists on an already fine-tuned retriever and after the pool adds tens of thousands of distractors.

Query Decomposition improves I3 under Per-Split but lowers the average for both tool representations, and under Full-Corpus it lowers every split. A retriever fine-tuned on complete query–tool pairs may simply be better aligned with original queries than with LLM-generated sub-queries. These results support retaining ToolRetriever’s original queries and strengthening its tool representation. With this configuration, the StableTool-

Bench agent’s Solvable Pass Rate rises from 60.13 to 64.14 (Appendix Table 14).

5.3 Transfer to TOOLRET and Downstream Task Completion

Per-Split rewards both sides of the framework, with Query Decomposition providing the larger share for the training-free encoders and CSF alone improving the supervised retriever. We therefore ask which side keeps its gain when the benchmark or the evaluation stage changes, with the systems of Table 2 where applicable.

Tool-side fusion. On TOOLRET with Qwen3, the only encoder we evaluate there, CSF without decomposition (CSF-no-QD in the appendix tables) gives the highest category-macro NDCG@5 among the first-stage methods, 35.29 against 32.27 for plain retrieval and 34.25 for Re-Invoke, with the gain coming from Web and Code while Custom falls below plain retrieval (Appendix Table 12). On the ToolBench Full-Corpus pool, where Re-Invoke also decomposes queries, full ToolFuse keeps the highest average NDCG@5 among the first-stage single-vector methods on all four encoders, by 1.4 to 3.8 points over Re-Invoke. Individual splits still favor other constructions, the reranker leads on GTE-large, and the tool-side change without decomposition falls below plain retrieval on mE5-large-instruct (Appendix C.6). The tool-side change on its own thus carries over on TOOLRET, while on Full-Corpus the advantage over Re-Invoke holds for the full pipeline rather than on every split.

Query Decomposition. Its benefit follows the query distribution. On TOOLRET it lowers Qwen3 from 35.29 to 33.38, below Re-Invoke, and on the supervised retriever it lowers the average under both pools (§5.2), although on the Full-Corpus pool it remains the larger of the two contributions on three of the four training-free encoders. The full pipeline’s Per-Split advantage therefore does not transfer unchanged, and the zero-shot reranker, which reads each candidate against the query, leads TOOLRET at 38.13.

Downstream task completion. With the retrieved top-8 tools passed to a GPT-4o-mini agent on StableToolBench, full ToolFuse raises query-weighted Solvable Pass Rate by 3.44–6.27 points across the four encoders; Qwen3 improves from 60.11 to 65.40, compared with 55.03 for the evalu-

ated hierarchical router. Aggregate retrieval gains thus accompany higher task completion in this agent setup, although I3-Inst declines on three encoders (Appendix C.9, Tables 15–16).

6 Analysis

This section asks how much tool-side vector fusion and query-side Query Decomposition each contribute, and what the per-tool fusion weight of CSF achieves relative to a searched fixed weight. All ablations and sensitivity analyses use the default backbone Qwen3-Embedding-0.6B and keep Query Decomposition except in w/o Query Decomp., so w/o intent view (73.50) is documentation-only retrieval with decomposition, not the plain baseline of Table 2 (64.72). Appendix D gives qualitative cases.

6.1 Component Ablation

Table 4 removes components one at a time. On the offline tool-representation side, w/o CSF (concat.) replaces vector fusion with Re-Invoke’s construction, concatenating each synthetic intent with the documentation, encoding the copies, and averaging their vectors, while keeping ToolFuse’s own generated intents and Query Decomposition. Average NDCG@5 drops from 75.49 to 72.27, with I3 losing 4.89 points, and since generated content and query processing are identical, the drop isolates tool-representation construction. On Qwen3 the concatenated representation (72.27) even falls below documentation alone (w/o intent view, 73.50), so on this encoder the fusion method decides whether the intent view helps at all. Removing either view degrades the average, so both views contribute, and vector fusion stays above concatenation on average for all four encoders (Appendix C.8).

On the online side, removing Query Decomposition lowers the average by 9.04 points, with changes of −0.66 on I1, −11.23 on I2, and −15.22 on I3, so its contribution is concentrated on the two multi-tool splits. The tool-side contribution, by contrast, holds under either query processing: the fused index adds about two points with or without decomposition on Qwen3 (§5.1) and +1.46 to +2.08 in each row pair of Table 3 on the supervised ToolRetriever.

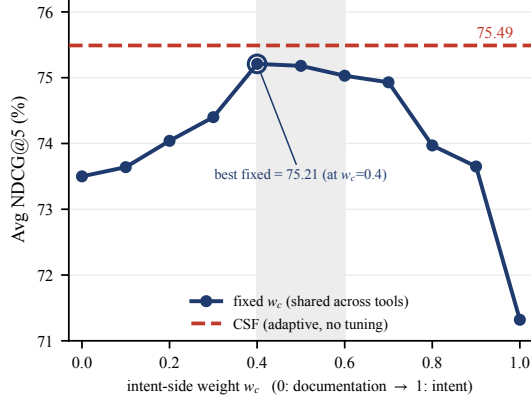


Figure 3: **Effect of the fusion weight (Per-Split Avg NDCG@5, %).** Blue: a shared intent-side weight w_c (0: documentation only; 1: intent only). Red: CSF, which sets weights without query labels. The best shared weight reaches 75.21; CSF reaches 75.49.

Variant	I1	I2	I3	Avg
ToolFuse	76.23	71.47	78.76	75.49
<i>Offline tool representation</i>				
w/o CSF (concat.)	74.25	68.69	73.87	72.27
w/o intent view	73.17	69.90	77.44	73.50
w/o documentation	73.02	68.70	72.24	71.32
<i>Online query processing</i>				
w/o Query Decomp.	75.57	60.24	63.54	66.45

Table 4: **Per-Split ablation (NDCG@5, %).** w/o CSF (concat.) replaces vector fusion with text-level concatenation on the tool side, holding all else fixed. Bold marks the best value per column.

6.2 Effect of the Fusion Weight

To probe the per-tool weight, we replace α_ℓ in (2) with a single global constant $w_c \in [0, 1]$ shared by all tools. Figure 3 plots that sweep, in increments of 0.1, alongside CSF, with per-split numbers in Appendix Table 9.

The best fixed fusion weight differs across the three ToolBench Per-Split pools: $w_c \approx 0.6$ on I1, 0.7 on I2, and 0.4 on I3. The best single shared value, $w_c = 0.4$, gives 75.21 average NDCG@5. Selecting the best value independently for each split gives an oracle average of 75.50. CSF uses corpus geometry rather than labeled queries to set per-tool weights and reaches 75.49; it matches or exceeds the best fixed weight on I2 and I3 and is within a third of a point on I1 (Appendix Table 9). Appendix C.4 gives the α_ℓ distribution.

Choice of the complementarity score. Scoring complementarity by overall dissimilarity instead of Eq. (1), with cosine distance ($1 - \cos$) between

the two views, gives a close average NDCG@5 (75.13 vs. 75.49, Appendix Table 9); we keep the orthogonal residual as the default for the geometric reading in §3.2.

Number of synthetic intents. Sweeping the intents generated per tool from $n = 1$ to 30 with the rest of the pipeline fixed leaves average NDCG@5 between 74.70 and 75.71 for every $n \geq 7$, so the default $n = 10$ (75.49) is not a finely chosen count. Appendix C.1 reports the sweep together with the two ways of combining the n intents into the intent view.

7 Conclusion

We presented **ToolFuse**, a training-free tool-retrieval framework that treats a tool’s documentation and its LLM-generated synthetic intent as two views rather than one text. Complementarity-based Semantic Fusion (CSF) encodes the views separately and weights them by how far a tool’s intent view extends beyond its documentation direction, read against the corpus distribution, while the query side splits a request that bundles several needs into focused sub-queries and aggregates their result lists by rank.

On ToolBench Per-Split, ToolFuse raises average NDCG@5 over documentation-only retrieval on all four dense encoders and has the highest average among the training-free systems we compare, and CSF reaches the accuracy of a per-split oracle weight sweep from corpus geometry alone. The two sides transfer differently. ToolFuse keeps an aggregate advantage of 1.4 to 3.8 points over Re-Invoke on the Full-Corpus pool, and CSF alone leads the first-stage methods on TOOLRET. As a representation-level construction, CSF also improves the fine-tuned ToolRetriever under both pools and on downstream task completion. Query Decomposition improves the Per-Split averages on the four dense encoders but lowers the average on TOOLRET and on the supervised retriever. Deciding when to invoke it and making the view weights query-aware without giving up the offline index are natural next steps.

Limitations

ToolFuse builds part of every tool representation from generated text that the tool’s author never wrote. When documentation is thin or ambiguous, the synthetic intent can describe a use the tool does not serve, and CSF weights the two views

by their geometry rather than by their correctness. Such an intent therefore enters the fused vector and, precisely because it sits far from the documentation direction, receives a substantial weight. Appendix D.4 traces a case where the intent view lowers the rank of a tool that documentation-only retrieval had already placed correctly. The framework also needs an LLM for both generation steps, and they differ in cost and in sensitivity: synthetic intent is generated once per tool offline and changing its generator moves average NDCG@5 within one point, whereas Query Decomposition runs on the path of every query and changing its model moves the average by up to about five points (Appendix C.10).

CSF fixes each tool’s fusion weight before any query arrives. That is what allows one vector per tool and an index built once offline, but the balance between the two views cannot follow the query: when a query already matches one view well, the other is still mixed in at its stored proportion and can dilute a signal that was correct on its own, and a misleading intent cannot be suppressed for the queries it harms while kept for those it helps. Query Decomposition has a matching weakness on the other side, since splitting is decided from the query text alone and can separate an atomic functional constraint from the need it qualifies, after which rank aggregation still admits candidates from the resulting sub-query. It is therefore a technique whose benefit depends on the query distribution rather than an always-on component, and a deployment would need a criterion for when to decompose, which we do not develop here.

Ethical Considerations

We evaluate on publicly released resources. ToolBench is distributed under the Apache 2.0 license for research use, and TOOLRET aggregates previously released datasets under their own licenses. The tool documentation we encode is the documentation these benchmarks distribute, and the synthetic intents we generate from it serve only as retrieval-side text. Both the offline intent generation and the online query rewriting call GPT-4o-mini through a commercial API, so tool documentation and benchmark queries are sent to a third-party provider under its terms of use. A deployment that decomposes real user queries would send those queries to the same kind of service. The method itself does not settle that data-handling de-

cision. Appendix A reproduces every prompt we used.

A synthetic intent is generated text rather than a description the tool provider wrote. It can overstate or misdescribe what a tool does, and because it enters the fused index it can move a tool up the ranking for requests it does not actually serve, as the Limitations section notes. Retrieval also sits upstream of execution. ToolFuse changes which tools an agent is shown, so a ranking error can propagate into the call the agent makes, and a retriever that surfaces the right tool more reliably surfaces harmful capabilities more reliably as well. We measure ranking quality and downstream Solvable Pass Rate, and neither of those quantifies the consequences of an executed call.

Acknowledgments

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

References

- Raviteja Anantha, Bortik Bandyopadhyay, Anirudh Kashi, Sayantan Mahinder, Andrew W. Hill, and Srinivas Chappidi. 2023. [ProTIP: Progressive tool retrieval improves planning](#). *arXiv preprint arXiv:2312.10332*.
- Francesco Bonchi, Carlos Castillo, Debora Donato, and Aristides Gionis. 2008. [Topical query decomposition](#). In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 52–60.
- Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. [M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.
- Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024b. [Re-Invoke: Tool invocation rewriting for zero-shot tool retrieval](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4705–4726.

- Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. 2009. [Reciprocal rank fusion outperforms condorcet and individual rank learning methods](#). In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759.
- Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. [Any-Tool: Self-reflective, hierarchical agents for large-scale API calls](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11812–11829. PMLR.
- Bowen Fang, Wen Ye, Yunyue Su, Jinghao Zhang, Qiang Liu, Yesheng Liu, Xin Sun, Shu Wu, Jiabing Yang, Baole Wei, and Liang Wang. 2026. [Tool-Weaver: Weaving collaborative semantics for scalable tool use in large language models](#). In *The Fourteenth International Conference on Learning Representations*.
- Edward A. Fox and Joseph A. Shaw. 1994. Combination of multiple searches. In *Proceedings of the Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, pages 243–252. National Institute of Standards and Technology.
- 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.
- Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. [StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156.
- Shui-Hsiang Hsu, Tsung-Hsiang Chou, Chen-Jui Yu, and Yao-Chung Fan. 2026. [STAR: Semantic table representation with header-aware clustering and adaptive weighted fusion](#). In *Proceedings of the ACM Web Conference 2026, WWW '26*, pages 8773–8776, New York, NY, USA. Association for Computing Machinery.
- Weize Kong, Swaraj Khadanga, Cheng Li, Shaleen Kumar Gupta, Mingyang Zhang, Wensong Xu, and Michael Bendersky. 2022. [Multi-aspect dense retrieval](#). In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3178–3186.
- Millicent Li, Tongfei Chen, Ben Van Durme, and Patrick Xia. 2025. Multi-field adaptive retrieval. In *The Thirteenth International Conference on Learning Representations*.
- Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.
- Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. 2024. Efficient and scalable estimation of tool representations in vector space. *arXiv preprint arXiv:2409.02141*.
- Rodrigo Nogueira and Jimmy Lin. 2019. [From doc2query to docTTTTTquery](#). Online preprint, University of Waterloo.
- Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*.
- Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, and 24 others. 2024a. [Tool learning with foundation models](#). *ACM Computing Surveys*, 57(4):1–40.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024b. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. [Towards completeness-oriented tool retrieval for large language models](#). In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1930–1940.
- Stephen Robertson and Hugo Zaragoza. 2009. [The probabilistic relevance framework: BM25 and beyond](#). *Foundations and Trends in Information Retrieval*, 3(4):333–389.
- 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.
- Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. [Retrieval models aren’t tool-savvy: Benchmarking tool retrieval for large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524.
- Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. [Multilingual E5 text embeddings: A technical report](#). *arXiv preprint arXiv:2402.05672*.
- Liang Wang, Nan Yang, and Furu Wei. 2023. [Query2doc: Query expansion with large language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9423.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025. ToolGen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations*.

Orion Weller, Kyle Lo, David Wadden, Dawn Lawrie, Benjamin Van Durme, Arman Cohan, and Luca Soldaini. 2024. [When do generative query and document expansions fail? A comprehensive study across methods, retrievers, and datasets](#). In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1987–2003, St. Julian’s, Malta. Association for Computational Linguistics.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. [Enhancing tool retrieval with iterative feedback from large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9609–9619.

Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. 2018. [Neural ranking models with multiple document fields](#). In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 700–708.

Shunyu Zhang, Yaobo Liang, Ming Gong, Daxin Jiang, and Nan Duan. 2022. [Multi-view document representation learning for open-domain dense retrieval](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5990–6000.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. [mGTE: Generalized long-context text representation and reranking models for multilingual text retrieval](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1393–1412, Miami, Florida, US. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. [ToolRerank: Adaptive and hierarchy-aware reranking for tool retrieval](#). In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16263–16273.

A Prompt Templates

We list the two prompt templates ToolFuse uses, where { . . . } marks variables filled in at run time. Both prompts use a few short few-shot examples

to control output format, tone, and granularity. The examples for offline synthetic intent generation use generic concepts such as a loan calculator and an eGFR calculator and are not drawn from ToolBench. The example queries for online Query Decomposition come from the training split so that they match real query style, and we write their subqueries by hand. None of the examples carry a tool label, and no evaluation query ever enters a prompt.

A.1 Synthetic Intent Generation (Offline)

The offline prompt takes the documentation of a single tool as input and produces a fixed number of user-style synthetic intents. The in-prompt examples set the style we ask for, including a concrete scenario, a purpose phrase, and concrete parameter values only when the tool actually exposes parameters.

Prompt template.

System:

Write queries that capture what a real person would search for when looking for a tool to handle this specific task. Each query should feel grounded in a concrete scenario and carry enough context -- the service name, data type, or action that makes this tool distinct -- so it reads like a meaningful search, not a generic request.

Where natural, end each query with a brief purpose phrase ("for X", "to Y") that anchors why someone needs this. Vary the style: some queries are short and keyword-dense, others are full goal-oriented sentences.

If the tool has no listed parameters, describe the general action only -- do not invent IDs, codes, or filter values.

Loan calculator:

"monthly payments for \$10,000 at 5% interest over 3 years for budget planning"
"Total interest on a 30-year \$300k mortgage for a refinancing decision."
"Biweekly amortization table for a car loan to compare repayment options."

eGFR calculator:

"eGFR for 45-year-old male creatinine 1.0 mg/dL for kidney function screening."
"CKD stage for creatinine 2.5 in a 60-year-old woman for pre-appointment check."
"Kidney function score from serum creatinine for nephrology referral."

Carrier list (no parameters -- describe the action, no invented values):

"List all supported shipping carriers from TrackingMore for logistics setup."

"Retrieve available courier options for carrier selection reference."

Output plain text, one query per line, no quotation marks.

User:

Generate {num_queries} queries for this tool. Each should describe a distinct real-world scenario -- mix short keyword-dense queries with longer goal-oriented ones, and add a brief purpose phrase where it fits naturally. If no parameters are listed, describe only the general action.

```
## Tool Information
{api_info}
```

Output exactly {num_queries} queries, numbered 1-{num_queries}:

A.2 Synthetic Intent Generation Examples

We give two offline generation examples below. The tool documentation is shown in a consistent field order: tool name, API name, description, required parameters, and optional parameters.

Example 1: parameterized streaming lookup.

Tool documentation excerpt:

Tool name: Streaming Availability

API name: Get Ultra

Description: Get the details of a show via IMDb or TMDb id, including the global streaming availability info.

Required parameters: none

Optional parameters: output_language, tmdb_id, imdb_id

Generated synthetic intents:

1. Find streaming availability for the latest season of 'Stranger Things' using its IMDb ID for easy access.
2. Get the TMDb ID for 'The Crown' to check which platforms it's available on for my viewing plans.
3. Show me where to stream 'Inception' globally with its IMDb ID for weekend movie night.
4. Retrieve the streaming options for 'The Marvelous Mrs. Maisel' using its TMDb ID to finalize my watchlist.
5. Check the streaming platform for 'Breaking Bad' by searching its IMDb ID for binge-watching.
6. Locate where I can watch 'The Godfather' using its TMDb ID for a classic movie marathon.
7. List all streaming services offering 'The Mandalorian' using its IMDb ID for a family viewing session.
8. Find streaming information for 'Euphoria' with its TMDb ID to decide if I should subscribe to a new service.
9. Get the global streaming availability for 'Game of Thrones' using its IMDb ID for my rewatch plans.

10. Check where 'Parasite' is streaming globally using its TMDb ID for an upcoming movie night with friends.

Example 2: authorization-status tool.

Tool documentation excerpt:

Tool name: Helioviewer v1

API name: checkYouTubeAuth

Description: Check to see if Helioviewer.org is authorized to interact with a user's YouTube account via the current browser session.

Required parameters: none

Optional parameters: callback

Generated synthetic intents:

1. Check if Helioviewer.org can access my YouTube account for solar event sharing.
2. Confirm YouTube authorization status for Helioviewer to upload my videos of solar observations.
3. Is Helioviewer authorized to interact with my YouTube account in this session for educational content?
4. Verify Helioviewer access to my YouTube for livestreaming solar events.
5. Check YouTube authentication for Helioviewer.org to ensure I can use features for sharing.
6. Determine if Helioviewer can post to my YouTube channel from this browser for educational outreach.
7. Is my YouTube account linked to Helioviewer.org for accessing archived solar images?
8. Check authorization for Helioviewer to access my YouTube account for presenting solar research.
9. Confirm if Helioviewer has permission for YouTube integration to enhance my astronomy projects.
10. Validate YouTube connection for Helioviewer.org to manage uploads of solar activity videos.

A.3 Query Decomposition (Online)

The online prompt takes the user query as input and returns a JSON list of sub-queries, with each sub-query corresponding to one type of tool that the query needs to retrieve.

Prompt template.

System:

Break the user query into sub-tasks -- one per distinct type of tool needed. Think of each sub-task as a search query someone would type to find exactly that tool.

The guiding question: "What different kinds of tools does this query need?" One sub-task per answer. If multiple intents would be handled by the same type

of tool, write one combined sub-task.

Style: concrete action verb + domain-specific object + brief purpose phrase ("for X", "to Y"). Preserve brand names, IDs, and service names exactly. Only include what the user explicitly asked for.

Example 1 -- two different tool types -> split
Input: "track my correo argentino shipment 987654321 and check if SQUAKE is operational"

Output: {"sub_tasks": [
 {"query": "Track Correo Argentino shipment 987654321 for current delivery status."},
 {"query": "Check SQUAKE service health and uptime for operational confirmation."}]}

Example 2 -- same tool type across multiple entities -> one sub-task

Input: "get hotel prices in Miami, Los Angeles, and Honolulu for July, plus EV charging stations near each"

Output: {"sub_tasks": [
 {"query": "Historical hotel prices in Miami, Los Angeles, and Honolulu for July trip planning."},
 {"query": "EV charging stations near Miami, Los Angeles, and Honolulu for travel routing."}]}

Example 3 -- one tool handles everything -> single sub-task

Input: "for order 9873458734, get the details, item list, and shipping status"

Output: {"sub_tasks": [
 {"query": "Order details, items, and shipping status for order ID 9873458734."}]}

Return only valid JSON: {"sub_tasks": [{"query": "..."}, ...]}

User:

Decompose into sub-tasks -- one per type of tool needed. Each sub-task should read like a standalone tool-search query with a brief purpose phrase. Merge anything a single tool type could handle.

Query: {query}

Return JSON only.

A.4 Query Decomposition Examples

We give two online Query Decomposition examples below.

Example 1: recommendation, availability, and food planning.

Original query:

I'm planning a hiking trip with my friends and we want to listen to some inspiring podcasts during the journey. Can you recommend some motivational podcasts and provide the streaming availability on platforms like Spotify and Apple Podcasts? Also, suggest some healthy snacks for the hike.

Decomposed sub-queries:

1. Recommend motivational podcasts for hiking trip inspiration.
2. Check streaming availability of recommended podcasts on Spotify and Apple Podcasts.
3. Suggest healthy snacks for hiking.

Example 2: cross-platform video metadata.

Original query:

I want to create a playlist of my favorite music videos from YouTube. Get the related tags for the 'Music' category from Vimeo to enhance the playlist. Also, provide me with information about the 'Music' category from Vimeo.

Decomposed sub-queries:

1. Create a playlist of favorite music videos from YouTube.
2. Get related tags for the 'Music' category from Vimeo to enhance the playlist.
3. Provide information about the 'Music' category from Vimeo.

A.5 Retrieval Query Instruction (Instruction-Tuned Encoder)

Following their official recommended usage, the instruction-tuned encoders Qwen3-Embedding and mE5-large-instruct prepend the task instruction prefix below before encoding a query, whereas the non-instruction-tuned BGE-M3 and GTE-large use no query instruction. This prefix acts on the query side only and leaves the encoding of tool documentation and synthetic intent unchanged. It is identical under the plain and ToolFuse conditions.

Instruct: Given a user's task request, retrieve API tool descriptions whose functionality satisfies the request
Query: {query}

B Experimental Details

B.1 Dataset Statistics

Table 5 lists the candidate-pool size and the number of test queries under each setting.

B.2 Hyperparameter Settings

Table 6 collects the default hyperparameters shared by all experiments unless a section states otherwise.

Setting	Corpus	# Tools	# Queries
Per-Split	$\mathcal{T}_{I1}$	10,439	600
	$\mathcal{T}_{I2}$	13,139	399
	$\mathcal{T}_{I3}$	1,603	100
Full-Corpus	$\mathcal{T}_{\text{full}}$	49,415	1,099

Table 5: ToolBench evaluation statistics. # Tools is the candidate-pool size and # Queries is the number of test queries. Under Full-Corpus, the same I1, I2, and I3 test queries (600, 399, 100; 1,099 total) are evaluated against the enlarged $\mathcal{T}_{\text{full}}$ pool, the unfiltered RapidAPI crawl released with ToolBench, rather than against a separately curated query set.

Symbol	Value
Encoder f	Qwen3-Embedding-0.6B (1024-d)
Intent-generation LLM	GPT-4o-mini
Query-decomposition LLM	GPT-4o-mini
# synthetic intents n	10
Fusion temperature τ	4
Per-sub-query retrieval L	50
Final ranked-list size K	5
Embedding similarity	cosine

Table 6: Hyperparameters used throughout the paper unless stated otherwise.

B.3 Models and Compute

Our pipeline is training-free: no encoder or LLM is fine-tuned, and all reported numbers are inference-time only. The four dense encoders are Qwen3-Embedding-0.6B (Qwen/Qwen3-Embedding-0.6B, our primary encoder, 0.6B parameters), BGE-M3 (BAAI/bge-m3, 0.57B), mE5-large-instruct (intfloat/multilingual-e5-large-instruct, 0.56B), and GTE-large (the Alibaba-NLP/gte-large-en-v1.5 checkpoint, 0.43B); the supervised ToolRetriever is a BERT-base bi-encoder (0.11B). Synthetic intent generation and Query Decomposition both use GPT-4o-mini through the OpenAI API, and the LLM-backbone ablation (Appendix C.10) additionally uses GPT-4.1 through the same API and the open-weight models listed there (Qwen3-8B/14B/32B for intent generation; Qwen3-30B-A3B, Gemma-4-26B, GPT-OSS-20B, Ministral-14B, and Llama-3.1-8B for Query Decomposition), served locally over an OpenAI-compatible interface.

Local inference uses a single NVIDIA RTX A6000 (48 GB) with PyTorch, while FAISS retrieval runs on CPU. Encoding and retrieval for the core ToolBench Per-Split runs across the four dense encoders and the supervised retriever, in-

cluding the re-encoding needed by the LLM-backbone ablation, use under three GPU-hours on this card. The open-weight LLMs in that ablation (Appendix C.10) were served locally, and their generation time is not included in this figure. GPT-4o-mini Query Decomposition is cached once for the 1,099 ToolBench test queries and reused across the corresponding non-LLM-ablation experiments. Generating the ToolBench intents and decompositions uses fewer than 5,000 OpenAI chat-completion requests and costs under USD \$5. These figures exclude the separately reported Full-Corpus, TOOLRET, reranking, and downstream runs.

C Additional Results

C.1 The Intent View: Number of Intents and Their Combination

Figure 4 sweeps the number of generated intents per tool from $n = 1$ to 30 with the rest of the pipeline fixed. A single intent already yields a usable intent view (73.42), and average NDCG@5 rises by about two points as further intents cover more of a tool’s usage contexts. The curve then flattens, staying between 74.70 and 75.71 from $n = 7$ through $n = 30$ with small fluctuations and no further sustained gain. The default $n = 10$ reaches 75.49, and the reported results therefore do not rest on a finely chosen intent count.

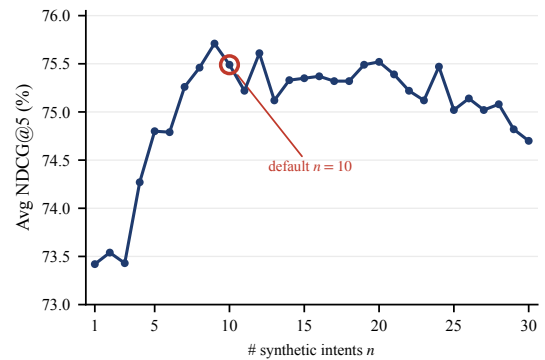


Figure 4: **Effect of the number of synthetic intents n (Per-Split Avg NDCG@5, %, Qwen3 + ToolFuse).** Accuracy rises from 73.42 at $n = 1$ and then flattens, staying between 74.70 and 75.71 from $n = 7$ through $n = 30$. The red circle marks the default $n = 10$.

§3.2 combines the n synthetic intents of a tool into the intent view $\mathbf{i}_\ell$ by concatenating them into a single text and encoding it once (concat-then-encode). We repeated the same sweep with the alternative that encodes each intent separately and av-

erages the n vectors (encode-then-average, $\mathbf{i}_\ell^{\text{avg}} = \frac{1}{n} \sum_{k=1}^n f(u_\ell^k)$), leaving the rest of the pipeline unchanged. Encode-then-average stays between 73.48 and 74.26 Per-Split Avg NDCG@5 across the sweep, while concat-then-encode covers 73.42 to 75.71; the two are close for $n \leq 3$ and differ by 0.92 to 1.70 points for $n \geq 9$. Table 7 gives the per-split comparison at the default $n = 10$, where the two reach 73.91 and 75.49 Avg NDCG@5.

Method ($n=10$)	I1	I2	I3	Avg
Encode-then-average	75.18	70.82	75.72	73.91
Concat-then-encode	76.23	71.47	78.76	75.49

Table 7: **Intent combination at the default $n = 10$ (Per-Split NDCG@5, %, Qwen3 + ToolFuse, rest of pipeline unchanged).** Concat-then-encode is the default intent combination (§3.2) and repeats the Qwen3 + ToolFuse row of Table 2. Bold marks the better value per column.

Method (I3, $n=10$)	@1	@3	@5
Encode-then-average	58.00	62.64	75.72
Concat-then-encode (default)	69.00	67.87	78.76

Table 8: **NDCG@ K on the I3 split at $n = 10$ (%).** Concat-then-encode leads by 11.00 points at NDCG@1, and the gap narrows as K grows but stays clear. Bold marks the better value per column.

Table 8 shows that on I3 at $n = 10$ the gap is 11.00, 5.23, and 3.04 points at NDCG@1, @3, and @5, so it concentrates at the strictest cutoffs. Both variants use the same user-style intents and differ only in whether the encoder processes their joint text or their vectors are averaged after separate encoding.

C.2 Full Fusion-Weight Sweep

Table 9 gives the full Per-Split NDCG@5 for the fixed-weight sweep of §6.2 (Figure 3), listing in its last two rows CSF driven by the two complementarity scores, the orthogonal residual and cosine distance ($1 - \cos$). The main-text figure plots only the Avg column.

C.3 Temperature Sensitivity and the Role of the Fusion-Weight Center

τ controls how steeply the z-score maps to the fusion weight α_ℓ . We set $\tau = 4$ a priori (§4) so that a tool whose complementarity deviates 3σ from the corpus mean still retains both views:

Weighting	I1	I2	I3	Avg
<i>Fixed intent-side weight</i>				
$w_c = 1.0$	73.02	68.70	72.24	71.32
$w_c = 0.9$	75.41	70.13	75.42	73.65
$w_c = 0.8$	75.14	70.65	76.12	73.97
$w_c = 0.7$	75.99	71.46	77.34	74.93
$w_c = 0.6$	76.55	71.22	77.31	75.03
$w_c = 0.5$	<u>76.45</u>	71.27	77.82	75.18
$w_c = 0.4$	<u>76.23</u>	70.89	<u>78.50</u>	<u>75.21</u>
$w_c = 0.3$	75.72	70.65	76.82	74.40
$w_c = 0.2$	75.33	69.86	76.94	74.04
$w_c = 0.1$	73.59	69.51	77.83	73.64
$w_c = 0.0$	73.17	69.90	77.44	73.50
<i>Adaptive tool-side weighting</i>				
CSF ($1 - \cos$)	76.19	71.02	78.19	75.13
CSF (orth. residual)	76.23	71.47	78.76	75.49

Table 9: **Effect of the fusion weight on retrieval (Per-Split NDCG@5, %).** The fixed weight w_c is the intent-side weight shared by all tools. CSF sets the weight automatically from a per-tool complementarity score; $1 - \cos$ uses cosine distance as an alternative complementarity score. The Avg column corresponds to Figure 3. Bold marks the best and underline the second-best value per column.

$\alpha_\ell = \text{sigmoid}(3/\tau) \approx 0.68$. Figure 5 reports the resulting sensitivity analysis.

Within the evaluated Qwen3 Per-Split setting, Avg NDCG@5 is insensitive to τ : $\tau = 1$ and the default $\tau = 4$ give 75.59 and 75.49, and the sweep from 1 to 32 spans 0.65 points. τ changes how widely the per-tool weights spread, but retrieval within this range does not depend on its exact value.

As τ grows, $z_\ell/\tau \rightarrow 0$ and every $\alpha_\ell \rightarrow 0.5$, so the per-tool weights converge on the same neutral value and the fused representation approaches an equal mix of the two views. In the limit $\tau \rightarrow \infty$ the procedure reduces to the fixed weight $w_c = 0.5$, which scores 75.18 (Table 9), 0.31 points below the default. The corpus-level standardization places $z_\ell = 0$ at the neutral weight 0.5, so the fusion weight is adjusted around the corpus’s average complementarity.

C.4 Verifying the α_ℓ Behavior of CSF

Figure 6 shows the distribution of the fusion weights α_ℓ that CSF assigns across all tools, complementing §6.2. The weights spread over $[0.19, 0.66]$ and center near 0.5, and about 40% of the tools receive an adjustment of $|\alpha_\ell - 0.5| > 0.05$ in one direction or the other. CSF therefore keeps the neutral weight as its center while shifting individual tools according to their complementarity relative to the corpus distribution, and it does so

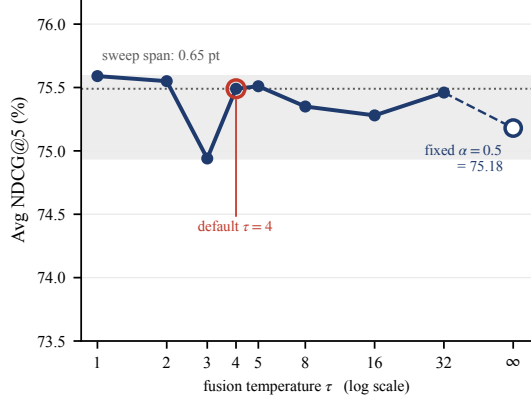


Figure 5: **Sensitivity to the fusion temperature τ (Per-Split Avg NDCG@5, Qwen3 + CSF + Query Decomposition).** Across τ from 1 to 32, Avg NDCG@5 varies by 0.65 points, and the default $\tau = 4$ lies within this plateau. The open marker takes $\tau \rightarrow \infty$, where every $\alpha_\ell \rightarrow 0.5$, so it coincides with the fixed weight $w_c = 0.5$ (Table 9, Avg 75.18).

without labels; a fixed weight would instead pin every tool at the same point.

The ToolBench Full-Corpus pool gives a mean of 0.50 and a standard deviation of 0.06, similar in center and spread to the Per-Split pool. The two candidate-pool constructions evaluated here thus yield similar marginal weight distributions, and the weights are always assigned per tool relative to the active pool.

C.5 Aggregation Strategies

Algorithm 1 gives the rank aggregation procedure used in §3.3. At each rank j , it considers the j -th candidate from every sub-query in descending similarity order, skips previously selected tools, and stops after K distinct tools.

Table 10 compares five aggregation strategies under the same Qwen3 Per-Split setting. The two rank aggregation variants differ only in the order of candidates at the same rank: the default sorts them by similarity, while the other omits the within-round sort. RRF (Cormack et al., 2009) combines reciprocal ranks across sub-queries, whereas the two CombSUM (Fox and Shaw, 1994) variants use the mean or sum of per-sub-query similarities. All other ToolFuse settings remain fixed.

The two rank aggregation variants achieve the highest averages. Similarity sorting raises average NDCG@5 from 75.14 to 75.49, improving I2 by 0.34 points and I3 by 0.83 while reducing I1 by 0.13. Its average contribution is modest; both

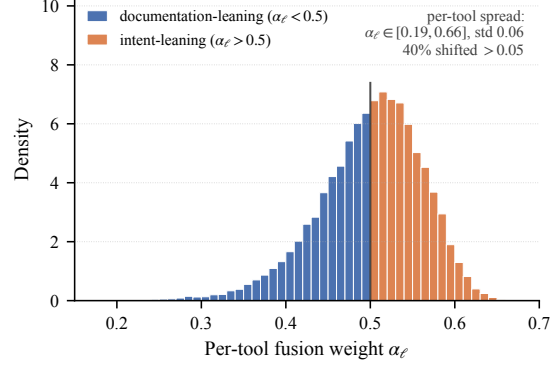


Figure 6: **Per-tool distribution of the fusion weight α_ℓ (Per-Split I1–I3 combined, $\tau = 4$, Qwen3).** Color distinguishes tools pushed toward documentation ($\alpha_\ell < 0.5$, blue) from those pushed toward intent ($\alpha_\ell > 0.5$, orange); the black line marks the neutral point 0.5 that a fixed weight would pin every tool to. The α_ℓ values spread over $[0.19, 0.66]$ rather than collapsing to a single weight.

Strategy	I1	I2	I3	Avg
Rank aggregation (w/ sort)	76.23	71.47	78.76	75.49
Rank aggregation (w/o sort)	76.36	71.13	77.93	75.14
RRF	72.06	67.73	65.01	68.27
CombSUM (mean)	73.79	69.35	79.04	74.06
CombSUM (sum)	73.75	67.22	69.21	70.06

Table 10: **Aggregation-strategy ablation (Per-Split NDCG@5, %, Qwen3 + ToolFuse).** Rank aggregation with within-round similarity sorting is the default and has the highest average. Bold marks the best per column.

variants preserve the priority of higher-ranked candidates from each sub-query. We retain similarity sorting as the default based on the three-split average.

CombSUM (mean) has the highest I3 result (79.04), but its average (74.06) trails both rank aggregation variants. CombSUM (sum) and RRF reach averages of 70.06 and 68.27, respectively. Both accumulate contributions when a candidate appears in multiple lists, but their different results and the stronger mean variant leave the cause of the gap unresolved. Explaining it would require measuring candidate overlap across the sub-query lists, which we leave open.

C.6 The Full-Corpus Candidate Pool

Table 11 repeats the five methods of Table 2 on the Full-Corpus pool of all 49,415 tools and adds CSF-no-QD, the fused tool representation queried without Query Decomposition, which is the w/o Query

Algorithm 1 Rank aggregation

Require: Per-sub-query ranked lists $R_1, \dots, R_m$ (each R_k a sequence of (tool, similarity) pairs); target size K

- 1: Output ranking $\mathcal{O} \leftarrow []$; selected set $\mathcal{S} \leftarrow \emptyset$
- 2: **for** $j = 1, 2, \dots, \max_k |R_k|$ **do**
- 3: $B_j \leftarrow \{(t_k^{(j)}, r_k^{(j)}) : 1 \leq k \leq m, j \leq |R_k|\}$
- 4: Sort B_j by similarity r from high to low
- 5: **for each** $(t, r) \in B_j$ (in sorted order) **do**
- 6: **if** $t \notin \mathcal{S}$ **then**
- 7: Add t to $\mathcal{S}$ and append t to the end of $\mathcal{O}$
- 8: **if** $|\mathcal{O}| = K$ **then**
- 9: **return** $\mathcal{O}$
- 10: **end if**
- 11: **end if**
- 12: **end for**
- 13: **end for**
- 14: **return** $\mathcal{O}$

Decomp. variant of Table 4 carried to this pool. ToolFuse has the highest macro-average NDCG@5 among the single-vector methods on all four encoders. The advantage is an aggregate one: individual splits still favor other constructions, with plain retrieval leading I1 and Re-Invoke exceeding ToolFuse on I3 on Qwen3, and Re-Invoke exceeding ToolFuse on GTE-large I2. The reranker has the highest average on GTE-large and the highest I3 score for all four encoders, whereas under Per-Split it led I3 on GTE-large alone, so its relative standing on I3 improves in the enlarged pool. The hierarchical router has a lower macro average than plain flat retrieval for every encoder in this pool as well. CSF-no-QD improves on plain retrieval in three of the four encoder averages, but lowers mE5-large-instruct from 44.99 to 44.79, including an I3 drop from 30.34 to 22.89. Thus, the tool-side change alone does not improve every Full-Corpus setting.

C.7 Transfer to TOOLRET

TOOLRET pools heterogeneous retrieval tasks covering web APIs, code functions, and customized applications, providing a distributional transfer test beyond ToolBench’s explicitly constructed tool-composition splits. We use Qwen3-Embedding-0.6B and report all retrieval metrics at cutoff 5. Table 12 reports category-level NDCG and aggregate metrics averaged over the three TOOLRET

categories. The methods follow Table 2 except for the hierarchical router, which the benchmark cannot host: TOOLRET is assembled from separate retrieval corpora and provides no native category tree of the kind the router descends on ToolBench.

CSF-no-QD improves aggregate NDCG from 32.27 for plain dense retrieval and 34.25 for Re-Invoke to 35.29, with corresponding gains on MAP, recall, precision, and completeness. Query Decomposition does not help under this transfer. Full ToolFuse reaches 33.38 aggregate NDCG, below CSF-no-QD. It raises Web NDCG slightly (41.70 to 42.18) but lowers Code (33.60 to 29.35) and Custom (30.56 to 28.63). Because the decomposition stage also rewrites a single-need query into one sub-query, the decline may stem from oversplitting, altered functional constraints, or the aggregation step; in either case the query-side procedure does not transfer unchanged to this query distribution.

The reranker is the strongest system in the table, as expected of a stage that reads each candidate against the query. It re-scores a first-stage list rather than replacing one, so we also ran it on top of each first stage in a separate NDCG-only pass. Aggregate reranked NDCG is 38.13 over plain dense retrieval, 39.51 over Re-Invoke, 39.54 over CSF-no-QD, and 38.86 over full ToolFuse. All three augmented first stages exceed the reranked plain baseline. The ordering of first stages is not fully preserved after reranking, and CSF-no-QD and Re-Invoke remain numerically close (39.54 vs. 39.51).

C.8 Component Ablation Across Encoders

Table 13 repeats the component ablation of §6.1 on all four dense encoders, removing one component at a time under the same Per-Split Query Decomposition pipeline. Since w/o intent view keeps documentation only, the gap from it to ToolFuse is the tool-side contribution that the intent view and CSF add. This contribution is positive on every encoder and ranges from +1.99 on Qwen3-Embedding-0.6B to +6.48 on GTE-large, so it is not specific to one backbone. The pattern seen on Qwen3 holds on all four: each view contributes to the average of every encoder, and removing Query Decomposition gives the largest drop on the multi-tool I2 and I3.

Across the four encoders the fused representation ranks above either single view alone in 11 of the 12 encoder-split cells. The exception is mE5-large-instruct on I3, where the intent view

Method	I1	I2	I3	Avg	Method	I1	I2	I3	Avg
<i>Qwen3-Embedding-0.6B</i>					<i>BGE-M3</i>				
plain	69.20	49.30	36.11	51.54	plain	54.27	44.85	14.31	37.81
Re-Invoke	62.67	62.70	40.26	55.21	Re-Invoke	63.67	62.51	38.42	54.87
Hierarchical [†]	52.95	43.22	38.01	44.73	Hierarchical [†]	25.93	23.15	16.43	21.84
Reranker [†]	63.29	55.03	49.45	55.92	Reranker [†]	62.73	53.67	47.00	54.47
CSF-no-QD	68.89	51.15	37.23	52.43	CSF-no-QD	61.48	51.27	32.01	48.25
ToolFuse	66.80	64.26	38.83	56.63	ToolFuse	64.91	62.94	44.86	57.57
<i>mE5-large-instruct</i>					<i>GTE-large</i>				
plain	60.95	43.68	30.34	44.99	plain	52.51	39.35	28.74	40.20
Re-Invoke	61.90	62.02	33.94	52.62	Re-Invoke	55.91	54.49	31.62	47.34
Hierarchical [†]	37.17	34.76	26.23	32.72	Hierarchical [†]	27.15	31.04	21.66	26.62
Reranker [†]	61.98	51.06	47.00	53.34	Reranker [†]	60.52	48.96	46.13	51.87
CSF-no-QD	64.12	47.35	22.89	44.79	CSF-no-QD	59.78	45.17	31.00	45.32
ToolFuse	65.47	62.26	38.61	55.45	ToolFuse	60.28	54.07	39.15	51.17

Table 11: **Complete training-free comparison on ToolBench Full-Corpus (NDCG@5, %).** The candidate pool contains all 49,415 tools. The five methods and their order follow Table 2; [†] again marks the two designs that change the retrieval procedure. CSF-no-QD is the extra row, the fused tool representation queried without Query Decomposition, that is, the w/o Query Decomp. variant of Table 4 under this pool. Results are not directly comparable to the Per-Split pool. Bold marks the best macro average within each encoder block, and the layout follows Table 13.

Method	Web	Code	Custom	NDCG	MAP	Recall	Precision	Comp.
plain	41.01	24.76	31.05	32.27	27.70	36.99	11.61	28.37
Re-Invoke	40.14	30.92	31.70	34.25	29.65	39.24	12.10	30.82
Reranker [†]	43.19	35.22	35.97	38.13	32.76	43.83	14.69	33.94
CSF-no-QD	41.70	33.60	30.56	35.29	30.62	40.14	12.18	31.56
ToolFuse	42.18	29.35	28.63	33.38	28.76	38.30	11.95	29.92

Table 12: **Training-free transfer to TOOLRET (% , all metrics at 5).** Web, Code, and Custom (customized apps) are category-level NDCG. Aggregate columns are macro averages over the three categories; Comp. is Completeness@5, the fraction of queries whose target tools all appear in the top five (Qu et al., 2024). Method names follow Table 2; [†] again marks the reranker, which re-scores the plain dense top 100 with Qwen3-Reranker-0.6B. CSF-no-QD is the extra row, the fused tool representation queried without Query Decomposition. Bold marks the best value per column.

reaches 80.69 and the fused representation reaches 77.25. This result is consistent with CSF’s query-independent limitation: its corpus-derived weight does not change according to which view best matches a particular query. Even in this cell, vector fusion ranks above text-level concatenation.

C.9 Downstream Task Completion: StableToolBench Solvable Pass Rate

We evaluate whether retrieval gains carry through to end-to-end task completion on StableToolBench (Guo et al., 2024). We feed retrieved candidate lists into a full tool-use pipeline and report two comparisons: CSF stacked onto the supervised ToolRetriever, and the complete training-free ToolFuse pipeline against each plain encoder. StableToolBench is a reproducible version of ToolBench that replaces unstable live RapidAPI calls with a cache and a virtual API server, and assigns a Solv-

able Pass Rate using an LLM judge.

Setup. The agent is GPT-4o-mini using depth-first search-based decision tree (DFSdT) (Qin et al., 2024b) reasoning with width 2, and the Solvable Pass Rate is judged by GPT-4.1-mini, averaged over 3 samples per query. The retrieval side reuses the encoder of the supervised ToolRetriever and takes the top-8 tools on the Per-Split candidate pool as the candidate function list handed to the agent. The baseline runs ToolRetriever with cosine retrieval directly on the documentation index, and + CSF stacks CSF onto the same ToolRetriever encoder, using the same setup as the main-text default (concat-then-encode intent view, orthogonal-residual score, $\tau = 4$), with the rest of the pipeline unchanged. The evaluation follows StableToolBench’s standard instruction, category, and tool subsets for I1/I2/I3, 765 queries in total.

Variant	I1	I2	I3	Avg
<i>Qwen3-Embedding-0.6B</i>				
ToolFuse	76.23	71.47	78.76	75.49
w/o CSF (concat.)	74.25	68.69	73.87	72.27
w/o intent view	73.17	69.90	77.44	73.50
w/o documentation	73.02	68.70	72.24	71.32
w/o Query Decomp.	75.57	60.24	63.54	66.45
<i>mE5-large-instruct</i>				
ToolFuse	74.80	70.45	77.25	74.17
w/o CSF (concat.)	73.42	69.20	74.60	72.41
w/o intent view	71.16	68.05	68.72	69.31
w/o documentation	72.37	68.15	80.69	73.74
w/o Query Decomp.	71.78	56.31	57.17	61.75
<i>BGE-M3</i>				
ToolFuse	73.84	70.68	77.04	73.85
w/o CSF (concat.)	73.92	70.21	72.86	72.33
w/o intent view	68.33	65.31	68.87	67.50
w/o documentation	69.48	66.45	74.11	70.01
w/o Query Decomp.	70.18	58.18	58.85	62.40
<i>GTE-large</i>				
ToolFuse	70.31	64.70	73.26	69.42
w/o CSF (concat.)	67.64	61.88	70.55	66.69
w/o intent view	62.66	57.62	68.53	62.94
w/o documentation	67.72	60.07	72.70	66.83
w/o Query Decomp.	68.52	51.09	59.13	59.58

Table 13: **Component ablation across four dense encoders (Per-Split NDCG@5, %)**. Each variant removes or replaces one component from full ToolFuse under Per-Split. Query Decomposition is retained except in the w/o Query Decomp. row. w/o intent view keeps documentation only, so its gap to ToolFuse is the tool-side contribution. Bold marks the best value per column within each encoder.

Split	# Queries	Tool-Retriever	+ CSF	Δ
I1-Cat	153	62.6	68.8	+6.2
I1-Inst	163	57.2	58.8	+1.6
I1-Tool	158	64.6	70.0	+5.4
I2-Cat	124	63.0	68.1	+5.1
I2-Inst	106	58.3	60.1	+1.8
I3-Inst	61	47.5	50.5	+3.0
Weighted avg	765	60.13	64.14	+4.01

Table 14: **Downstream task Solvable Pass Rate (StableToolBench, %; bold marks the higher value)**. The agent is GPT-4o-mini + DFSDT (width 2), and the judge is GPT-4.1-mini (averaged over 3 samples per query). The retrieval side takes the top-8 tools under Per-Split and hands them to the agent; + CSF stacks CSF onto the same ToolRetriever encoder. # Queries is the number of test queries per subset. Δ is the change of + CSF over ToolRetriever.

Table 14 shows that switching the candidate list handed to the agent from ToolRetriever alone to ToolRetriever + CSF raises the weighted-average Solvable Pass Rate from 60.13 to 64.14 (+4.01), with positive changes on all six subsets. The largest changes occur on I1-Cat, I1-Tool, and I2-Cat. Solvable Pass Rate is judged by GPT-4.1-mini over three samples per query, so individual subset differences include judging and sampling variability. This table isolates CSF on top of the supervised ToolRetriever; Table 15 evaluates the complete training-free pipeline.

We next replace the supervised retrieval stage with each training-free dense encoder, comparing its plain documentation retrieval with full ToolFuse. Table 15 shows that ToolFuse raises the

query-weighted average Solvable Pass Rate for every encoder, by +3.44 to +6.27 points. Gains are not uniform across subsets: I3-Inst declines on three encoders, and mE5 also declines on I1-Tool. NDCG@5 and top-8 agent Solvable Pass Rate evaluate different stages, so a candidate-list change that improves aggregate retrieval can still displace a tool useful to the downstream agent on an individual subset.

Table 16 then places the hierarchical router against flat retrieval end to end. The router descends ToolBench’s native categories with the same Qwen3 encoder as the two flat systems, and we ran it on that encoder alone. Its comparison with plain retrieval tests this routing procedure; full ToolFuse additionally changes tool representation and query processing. The router reaches 55.03, below 60.11 for plain flat retrieval and 65.40 for ToolFuse, so its lower average retrieval score is accompanied by a lower task-completion score. Its largest loss is on I3-Inst (29.51 against 42.08), the I3 instruction subset. Other routing designs may behave differently. The reranker consumes a first-stage list rather than replacing one and was not run end to end.

C.10 Robustness to the Generating LLM

ToolFuse uses an LLM in two places: offline synthetic intent generation and online Query Decomposition (both use GPT-4o-mini in the main results). We replace the model in each role separately while keeping the remaining pipeline and hyperparameters fixed.

Replacing the synthetic intent generator. Table 17 replaces the offline synthetic intent generator

Encoder	Method	I1-Cat	I1-Inst	I1-Tool	I2-Cat	I2-Inst	I3-Inst	Avg	Δ
Qwen3	plain	64.16	63.50	68.35	51.48	57.23	42.08	60.11	–
	ToolFuse	72.77	64.52	71.84	60.48	65.41	42.62	65.40	+5.29
BGE-M3	plain	67.65	57.57	70.04	49.60	57.55	52.46	60.46	–
	ToolFuse	69.61	68.00	70.78	64.78	68.87	45.90	66.73	+6.27
mE5	plain	65.36	59.61	67.41	52.02	57.08	41.26	59.33	–
	ToolFuse	71.79	63.60	66.67	56.45	64.78	37.16	62.77	+3.44
GTE	plain	65.69	60.12	64.35	47.04	58.33	47.54	58.74	–
	ToolFuse	70.81	66.16	68.04	64.78	61.32	39.34	64.45	+5.71

Table 15: **End-to-end StableToolBench Solvable Pass Rate with training-free retrievers (%)**. Avg is query-weighted across the six solvable subsets. Δ is the change from the plain encoder within the same block. Table 16 reports the hierarchical router separately. Bold marks the best average within each encoder block.

Method	I1-Cat	I1-Inst	I1-Tool	I2-Cat	I2-Inst	I3-Inst	Avg
Hierarchical	61.11	57.67	62.76	52.15	48.74	29.51	55.03
plain	64.16	63.50	68.35	51.48	57.23	42.08	60.11
ToolFuse	72.77	64.52	71.84	60.48	65.41	42.62	65.40

Table 16: **Hierarchical routing against flat retrieval end to end (StableToolBench Solvable Pass Rate, %)**. All three systems use Qwen3-Embedding-0.6B and the setup of Table 15, holding the encoder fixed; ToolFuse additionally changes tool representation and query processing. Avg is query-weighted across the six subsets.

from GPT-4o-mini with three open-weight Qwen3 models (8B / 14B / 32B), keeping Query Decomposition fixed to GPT-4o-mini. Their Avg NDCG@5 ranges from 74.84 to 75.93, within one point of the GPT-4o-mini reference (75.49). Thus, all four generators give similar aggregate retrieval quality in this setting.

Intent-gen LLM	I1	I2	I3	Avg.
GPT-4o-mini [†]	76.23	71.47	78.76	75.49
Qwen3-8B	77.12	71.53	75.87	74.84
Qwen3-14B	76.75	71.62	79.41	75.93
Qwen3-32B	77.25	71.44	76.93	75.21

Table 17: **Replacing the synthetic intent generator (Per-Split NDCG@5, %, Qwen3 encoder + full ToolFuse)**. Only the offline intent-generation LLM changes; Query Decomposition is fixed to GPT-4o-mini, and the rest of the pipeline and hyperparameters are unchanged. [†] is the main-results reference (Table 2).

Replacing the Query Decomposition model. Table 18 fixes synthetic intent generation to GPT-4o-mini and swaps only the online Query Decomposition model, repeating the swap on all four encoders with one closed-source alternative (GPT-4.1) and five open-weight models (Qwen3-30B-

A3B, Gemma-4-26B, GPT-OSS-20B, Ministral-14B, Llama-3.1-8B). Most alternatives are within one to three points of the GPT-4o-mini Avg NDCG@5, while the largest gaps are about five points. Differences appear on all three splits and are largest on I3 for three of the four encoders. The prompt template in Appendix A.3 uses in-context examples to specify the requested output granularity, and the results show that decomposition quality varies across generators.

Across these replacements, offline intent generators vary within roughly one point in average NDCG@5. Query Decomposition variants show larger differences across all four encoders.

D Additional Case Studies

This appendix presents four case studies: a contrast between separate-view vector fusion and text-level concatenation, two cases that examine CSF’s weight behavior, and one failure-mode diagnosis. All cases use the online pipeline of this paper. The query first passes through Query Decomposition into sub-queries, each sub-query retrieves over the same candidate pool, and the results are aggregated by rank position. Unless stated otherwise, the compared settings share the same set of sub-query embeddings and the same aggregation strategy, and differ only in how the tool index is represented: documentation view only, intent view only, text-level concatenation, fixed-weight vector fusion, and full CSF. The fixed-weight row uses the best fixed intent-side weight of the split the case belongs to, $w_c = 0.6, 0.7$, and 0.4 for I1, I2, and I3, following the sweep in §6.2. The per-tool α_ℓ distribution spreads over $[0.19, 0.66]$ rather than collapsing to a single value, as Appendix C.4 reports. These cases illustrate local mechanisms and sources of failure.

Decomposition LLM	I1	I2	I3	Avg.
<i>Qwen3-Embedding-0.6B</i>				
GPT-4o-mini [†]	76.23	71.47	78.76	75.49
GPT-4.1	77.22	70.12	74.45	73.93
Qwen3-30B-A3B	75.88	69.87	77.73	74.49
Gemma-4-26B	75.41	69.42	77.65	74.16
GPT-OSS-20B	74.91	68.60	78.04	73.85
Ministral-14B	72.11	67.69	77.82	72.54
Llama-3.1-8B	71.83	67.67	78.68	72.73
<i>mE5-large-instruct</i>				
GPT-4o-mini [†]	74.80	70.45	77.25	74.17
GPT-4.1	74.25	69.17	77.61	73.68
Qwen3-30B-A3B	73.34	69.41	76.90	73.22
Gemma-4-26B	73.44	69.18	72.82	71.81
GPT-OSS-20B	73.30	69.61	75.34	72.75
Ministral-14B	72.33	68.08	76.31	72.24
Llama-3.1-8B	72.09	69.28	77.89	73.09

Decomposition LLM	I1	I2	I3	Avg.
<i>BGE-M3</i>				
GPT-4o-mini [†]	73.84	70.68	77.04	73.85
GPT-4.1	73.48	69.44	74.55	72.49
Qwen3-30B-A3B	72.50	70.11	74.10	72.24
Gemma-4-26B	73.15	69.29	71.57	71.34
GPT-OSS-20B	72.32	69.73	73.37	71.81
Ministral-14B	69.71	67.33	70.23	69.09
Llama-3.1-8B	70.00	67.13	74.13	70.42
<i>GTE-large</i>				
GPT-4o-mini [†]	70.31	64.70	73.26	69.42
GPT-4.1	70.61	63.27	66.17	66.68
Qwen3-30B-A3B	69.48	63.03	70.69	67.73
Gemma-4-26B	69.50	64.62	67.62	67.25
GPT-OSS-20B	68.54	64.63	70.93	68.03
Ministral-14B	68.23	63.33	71.72	67.76
Llama-3.1-8B	67.93	62.17	68.21	66.10

Table 18: **Replacing the Query Decomposition model across four dense encoders (Per-Split NDCG@5, %).** Only the online Query Decomposition LLM changes; synthetic intent generation is fixed to GPT-4o-mini, and the rest of the pipeline and hyperparameters are unchanged. [†] marks the GPT-4o-mini default used in the main results; its per-encoder row is the full-ToolFuse reference from Table 13.

D.1 Case A: Vector Fusion vs. Text-Level Concatenation

This case uses an I3 multi-intent query to contrast ToolFuse’s separate-view vector fusion against the text-level concatenation ablation of §6.1. The two variants use the same encoder, the same prompts, the same Query Decomposition, and the same aggregation strategy, and differ only in whether the tool representation encodes documentation and synthetic intent separately and then applies vector fusion, or concatenates the two into a single text and encodes it once.

Query. “I want to learn more about wildlife photography. Provide me with the category info for ‘photography’, the channel info for ‘National Geographic’, and the comments for the video with ID ‘901234’. Also, find a website where I can stream or download a tutorial video on wildlife photography...”

The query corresponds to four ground-truth endpoints: **Vimeo** / GetCategoryInfo, **Vimeo** / GetChannelInfo, **Vimeo** / GetVideosComment, and **YTStream** / Download.

Walkthrough. Table 19 identifies the missed endpoints. The two endpoints that text-level concatenation misses, **Vimeo** / GetCategoryInfo and **Vimeo** / GetChannelInfo, are highly similar at the documentation level to the sister endpoints that get selected. GetCategoryInfo has an empty API de-

scription and required parameters category and format, while GetRelatedChannels likewise has an empty description and the same parameters, and GetRelatedPeople shares a similar category-style interface. The concatenated representation ranks the sister endpoints above these two targets. In contrast, separate encoding keeps the intent view’s task-centric phrasing, such as “category info” and “channel info,” from being averaged with the near-identical schema text of the sister endpoints; Query Decomposition retrieves over these separate subqueries, and the full pipeline recalls all four ground-truth endpoints.

D.2 Case B: Intent-Leaning Tool

The second case examines a tool whose documentation leans toward parameter syntax and lacks user-task language: **MoviesDatabase** / /titles/x/titles-by-ids (ToolBench document id, docid, 5779).

Documentation. The endpoint’s functional description is only “Titles by ids list,” and the rest of the content is mainly the API parameter structure: the required parameter idsList (“Imdb id’s comma separated → tt0001702,tt0001856 ...”) and the optional parameters list and info (such as base_info / mini_info). The documentation almost entirely describes the calling format and says little about how a user would phrase “look up movie information” in natural language.

Rank	ToolFuse: separate-view vector fusion	Text-level concatenation
1	Vimeo / GetVideosComment ✓	Vimeo / GetVideosComment ✓
2	YTStream / Download ✓	YTStream / Download ✓
3	Vimeo / GetChannelInfo ✓	Vimeo / GetRelatedChannels
4	Vimeo / GetCategoryInfo ✓	Vimeo / GetRelatedPeople
5	Social Media Data TT	Social Media Data TT
Hit@5	4/4	2/4

Table 19: **Top-5 comparison for Case A.** ToolFuse recalls all four ground-truth endpoints, while text-level concatenation replaces two of them with functionally similar Vimeo sister endpoints. Hit@5 counts ground-truth endpoints in the top five.

Tool representation	Rank of docid 5779
Documentation view only	31
Intent view only	3
Text-level concatenation	6
Fixed-weight fusion ($w_c=0.6$)	5
ToolFuse (CSF, $\alpha_\ell \approx 0.54$)	3

Table 20: **Ground-truth rank for the intent-leaning case.** Lower rank is better; bold marks the best.

Tool representation	Rank of docid 2380
Documentation view only	9
Intent view only	13
Text-level concatenation	12
Fixed-weight fusion ($w_c=0.6$)	11
ToolFuse (CSF, $\alpha_\ell \approx 0.44$)	10

Table 21: **Ground-truth rank for the documentation-leaning case.** Lower rank is better; bold marks the best.

Synthetic intent. One LLM-generated synthetic intent is: “Request information for IMDb IDs tt1399683, tt2452046, and tt0457430 from Movies-Database, collecting base information for thorough analysis of each title.”

Query. “Can you provide me with the details of the movies with the IMDb IDs tt1234567, tt9876543, and tt2468024? Additionally, give me the crew details and additional information for each movie.”

In Table 20, documentation view only ranks the endpoint at 31; its text is dominated by parameter syntax, whereas the query asks for “movie details / crew details / additional information.” Intent view only ranks it at 3, and CSF assigns $\alpha_\ell \approx 0.54$ toward that view. The fused representation also ranks it at 3, ahead of fixed-weight vector fusion and text-level concatenation in this case.

D.3 Case C: Documentation-Leaning Tool

The third case examines a tool whose documentation is already close to user query language: **Body Mass Index (BMI) Calculator** / Weight Category (docid 2380).

Documentation. The endpoint’s functional description is “Retrieve the weight category of an individual based on their BMI,” with a bmi parameter and a short return schema (bmi / weightCategory). This documentation already expresses the usage context directly.

Synthetic intent. The corresponding synthetic intents are mostly rewrites that plug in specific values, such as: “Identify the weight category for an individual using the Body Mass Index (BMI) Calculator with a BMI value of 23.8.” These intents overlap heavily with the documentation and add little complementary information beyond it.

Query. “I’m designing a mobile app for tracking body mass index, and I need to integrate the BMI calculation API. Could you provide me with the necessary API endpoints and parameters?”

In Table 21, documentation view only gives the best rank (9). CSF places slightly more weight on documentation ($\alpha_\ell \approx 0.44$) and ranks the target at 10, compared with 11 for fixed-weight vector fusion, 12 for text-level concatenation, and 13 for intent view only. The one-rank gap from documentation-only illustrates that a query-independent fusion weight can still dilute the stronger view.

D.4 Case D: Failure Mode

We manually inspect the 196 I1 Per-Split queries on which ToolFuse’s rank-1 tool falls outside the ground-truth set, 32.7% of the 600 queries, consistent with the I1 NDCG@1 of 67.33 in Table 2. The inspection considers the best rank of the ground-truth tool under each semantic view and the category and tool-family relationship between the erroneous top-1 and the ground truth. Three sources of error recur.

Tool representation	Rank of docid 2314
Documentation view only	1
Intent view only	17
Text-level concatenation	1
Fixed-weight fusion ($w_c=0.6$)	2
ToolFuse (CSF, $\alpha_\ell \approx 0.46$)	2

Table 22: **Ground-truth rank for the fusion-error case.** Lower rank is better; bold marks the best.

- **Query Decomposition error:** the LLM-generated sub-query distorts the original request, for example by over-splitting a single-tool need or adding a functional constraint the query did not explicitly ask for.
- **Fusion error:** the ground-truth tool already ranks first when retrieved with the documentation view or the intent view alone, but is diluted by the weaker view after vector fusion and drops out of the top-1.
- **Ground-truth ambiguity:** the erroneous top-1 is functionally equivalent to the ground truth or belongs to the same tool family, but ToolBench labels only one of the endpoints as the ground truth.

Many failures involve more than one of these sources, and others are near-misses or mixed errors whose primary cause cannot be identified from the ranking alone. We therefore do not treat the three sources as mutually exclusive categories or estimate their proportions; a single failure need not be attributable to a single component. The following case presents one identifiable representation-fusion problem.

Representative fusion error. Query 87035 asks for the latest oil prices and a picture of today’s prices for “Gasohol E85” and “Hi Premium 97 Gasohol 95,” with the ground-truth endpoint being **Oil Thai Price** / Get latest price (docid 2314).

Table 22 shows that documentation view only ranks the ground truth at 1, while intent view only ranks it at 17. The generated intent uses the more abstract phrases “current market rates” and “visual reference” instead of the oil-product names present in the query and documentation. After fusion, the ground truth falls to rank 2, behind **Gas Price** / europeanCountries. Fixed-weight fusion and CSF ($\alpha_\ell \approx 0.46$) both produce rank 2, locating this example at representation fusion rather than at CSF’s weight variation: adding a weak intent view

dilutes a documentation view that already retrieves correctly.