From One Query to Many Tools: Latent Demand Decomposition for Multi-Tool Retrieval

Anonymous authors
Paper under double-blind review

Abstract

Modern agentic systems depend on selecting relevant tools before a language model can execute a task. As tool registries grow, selecting a small candidate set becomes a retrieval problem. However, single-embedding retrievers can struggle when a query contains multiple semantically distinct demands, as a single query representation may not align closely with every required tool. We introduce **LatDec** (Latent Demand Decomposer), a retrieval architecture that maps a query to multiple latent demand representations in a single encoder forward pass. Its independent heads are trained using Hungarian matching to align each representation with a distinct target tool. At inference, Confidence-Proportional Head Budgeting (CPHB) converts each head's confidence into its share of the retrieval budget. As a complementary signal, LatDec constructs a confidence-weighted population vector from all head representations, blends it with the frozen base query embedding, and combines the resulting ranking with the CPHB ranking using reciprocal rank fusion. We evaluate LatDec on two benchmarks: CapCorpus, a 22K-tool MCP retrieval benchmark that we construct and validate through independent human judgment, and the independently developed LiveMCPBench benchmark. On CapCorpus, LatDec substantially outperforms the frozen Sentence Transformer, matched bi-encoder, and multi-vector baselines. On LiveMCPBench, the benchmark's current sample size does not support a clear distinction between the full LatDec model and the baselines. Within LatDec, however, adding embedding-space fusion increases retrieval performance over CPHB alone by 26.3% relative, corresponding to an absolute Recall@10 improvement of 0.0797 (95% CI $[0.0445, 0.1172]$). In a quartile analysis, the gain is larger for queries whose confidence-weighted population vector is least aligned with the base query embedding, highlighting the importance of balancing demand-specific specialization at the head level with preservation of the query's overall semantics at the population level.

1 Introduction

Modern agentic systems solve complex tasks using external tools and APIs (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024), but as tool registries grow into the tens of thousands, exposing every tool description to the language model becomes impractical: token cost scales with the number of servers, tools per server, and tokens per schema Fan et al. (2025). The practical response is retrieval: embed the query and each tool description, rank by similarity, and expose only the top-K candidates, a pattern underlying systems from prompt-level scaffolding Lumer et al. (2025d) to LangGraph's bigtool.[1]

Most such retrievers assume a request can be represented by one semantic vector, an assumption that breaks down when a task expresses several distinct demands. Consider *"Debug why the authentication service is slow, open an incident ticket, and notify the on-call en-*

[1] `https://github.com/langchain-ai/langgraph-bigtool`

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

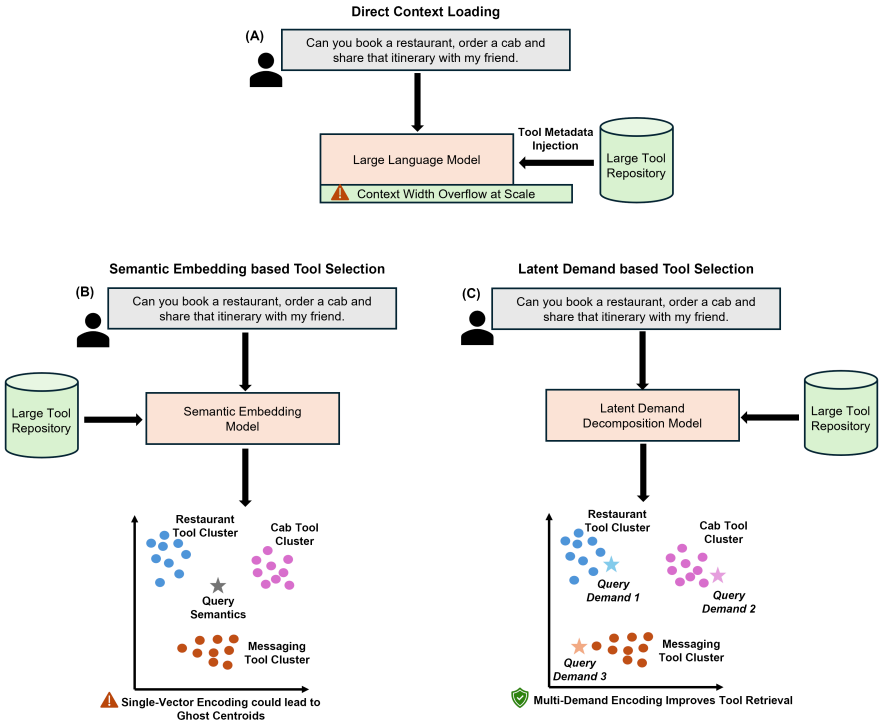

Figure 1: Motivation for latent demand decomposition. (A) Directly injecting metadata from a large tool repository into the language-model context becomes impractical as the repository grows. (B) A single embedding of a multi-demand query may favor one tool cluster or occupy a compromise region between clusters, producing the *ghost centroid* failure mode. (C) LatDec instead produces multiple demand-specific representations in a single encoder forward pass, allowing different heads to align with distinct tool clusters before retrieval.

gineer": satisfying it requires log analysis, issue tracking, and notification simultaneously, capabilities occupying semantically distant regions of the embedding space. This is not a rare edge case: multi-tool tasks average 2–5 required tools across several recent MCP and tool-retrieval benchmarks Mo et al. (2025); Shi et al. (2025); Lumer et al. (2025b); Cho et al. (2026). Compressing these heterogeneous demands into a single representation creates a single-vector bottleneck: the embedding may favor one demand at the expense of others or settle into a compromise region between them. We refer to the latter failure mode as a *ghost centroid*, where the query representation lies between semantically distant target tools without aligning closely with each one individually (Figure 1). Recent work is consistent with this being a representational limitation rather than one of scale or embedding quality alone Shi et al. (2025); Lumer et al. (2025b); Wang et al. (2026).

To address this gap, we introduce LatDec (Latent Demand Decomposer), which reframes multi-tool retrieval as learning a set of latent demand representations rather than compressing an entire request into a single vector. By representing distinct capability demands separately within one encoder forward pass, LatDec can retrieve from multiple regions of the tool-embedding space without relying on an auxiliary LLM to decompose the query at inference.

2 Related Work

One line of work uses external generative processing to interpret a query or guide tool discovery before or during retrieval: Tool-to-Agent Retrieval pre-splits queries into substeps Lumer et al. (2025c); ScaleMCP prompts an agent to issue one retrieval call per inferred sub-need Lumer et al. (2025b); MCP-Zero iteratively generates structured tool

requests as gaps are discovered mid-reasoning Fei et al. (2025); RAG-MCP selects tools via retrieval-augmented generation to limit the volume of tool metadata supplied to the language model Gan & Sun (2025); AnyTool uses hierarchical and iterative API discovery and selection Du et al. (2024); and Re-Invoke augments tool descriptions with synthetic queries during indexing and extracts tool-related intents from user queries for multi-view ranking Chen et al. (2024). These methods perform query interpretation or iterative tool discovery outside the learned retriever representation, introducing an additional generative call before or during retrieval.

Multi-vector and multi-view representations have a longer history in retrieval. Poly-encoders use multiple learned context codes for efficient attention-based candidate scoring Humeau et al. (2020). Luan et al. (2021) study sparse, dense, and attentional representations for text retrieval, and Zhang et al. (2022) learn multiple document-side views for open-domain dense retrieval. These works demonstrate the broader value of multi-representation retrieval, but serve different objectives: Poly-encoders use learned context codes for efficient candidate interaction, while the methods of Luan et al. and Zhang et al. enrich document-side representations. They do not specifically learn query-side heads to separate the distinct tool demands expressed by a multi-tool query.

Other work addresses complementary aspects of tool retrieval: ToolGen fuses tools into the LLM's vocabulary as generative tokens Wang et al. (2025); Graph RAG-Tool Fusion models inter-tool dependencies via a knowledge graph Lumer et al. (2025a); ToolRet fine-tunes embedding models on a large tool-retrieval set Shi et al. (2025); SkillRet shows domain-specific fine-tuning of a bi-encoder yields large gains on multi-capability queries Cho et al. (2026), though over Anthropic Agent Skills rather than tools exposed through MCP (Anthropic, 2024); and R3-Skill's compatibility-aware reranker corrects joint-retrieval errors after the fact Wang et al. (2026). LatDec differs in learning multiple query-side demand representations jointly within a single retriever forward pass and resolving their latent assignment to target tools during training.

3 The LatDec Architecture

3.1 Problem Formulation

Let $\mathcal{S} = \{s_1, \ldots, s_N\}$ be a corpus of N tools. Given a natural language query q, the goal is to retrieve the subset $T^* \subseteq \mathcal{S}$ of tools required to answer it, where $|T^*| \geq 1$. Both queries and tools are initially embedded by a frozen Sentence Transformer (ST, all-MiniLM-L6-v2) into $\mathbb{R}^{384}$. The central challenge is to retrieve tools corresponding to multiple semantically distinct demands without compressing the query into a single representation that may underrepresent some of them.

3.2 Architecture

Figure 2 shows the full model. The key design choice is *asymmetry*: only the query representation is learned and decomposed, while the tool side uses frozen ST embeddings directly, with no projection or learned parameters. This isolates learning to query decomposition while preserving the pretrained tool representation space. New tools can therefore be embedded and added to the index without retraining tool-side parameters.

Query backbone. A shared MLP (two blocks of Linear $\rightarrow$ LayerNorm $\rightarrow$ ReLU $\rightarrow$ Dropout: 384$\rightarrow$256, p=0.2, then 256$\rightarrow$256) maps the query ST embedding $\mathbf{e}_q \in \mathbb{R}^{384}$ to a backbone representation:

$$\mathbf{h} = \text{Backbone}(\mathbf{e}_q) \in \mathbb{R}^{256} \tag{1}$$

Head projections. Each of the H heads independently projects $\mathbf{h}$ into the same 384-dimensional space as the frozen tool embeddings (Linear(256$\rightarrow$384) $\rightarrow$ ReLU $\rightarrow$ Dropout(p=0.1) $\rightarrow$ Linear(384$\rightarrow$384)), sharing a single precomputed tool index at inference:

$$\hat{\mathbf{q}}_k = \text{Head}_k(\mathbf{h}) \in \mathbb{R}^{384}, \quad k = 1, \ldots, H \tag{2}$$

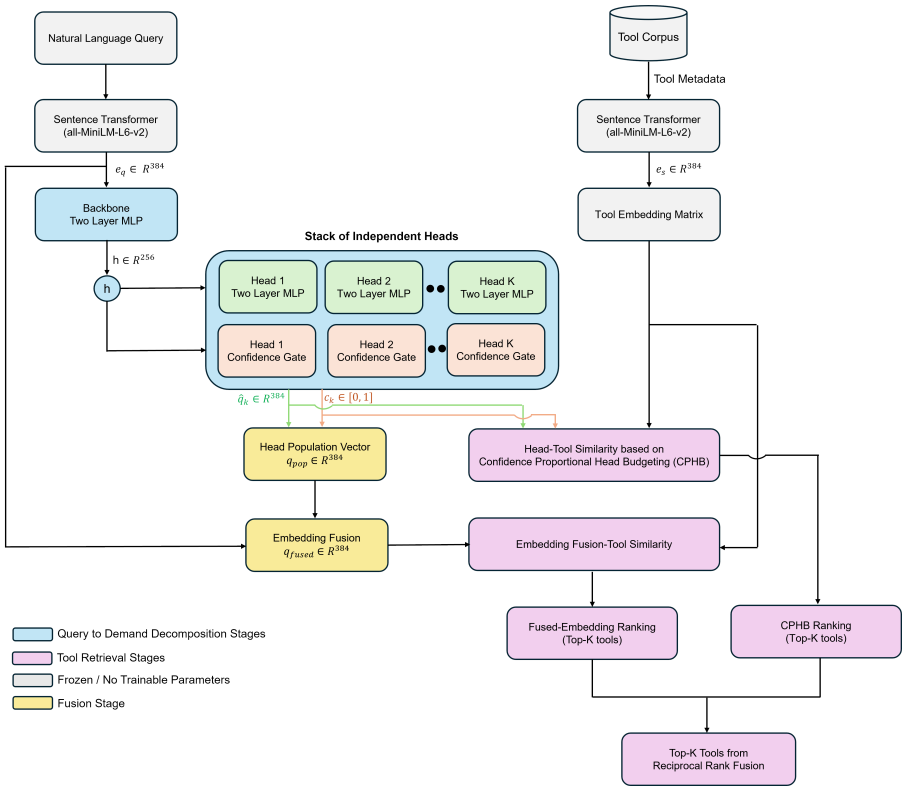

Figure 2: LatDec architecture. A frozen Sentence Transformer encodes both query and tools. The query backbone feeds H parallel heads, each producing a demand-specific projection $\hat{\mathbf{q}}_k$ and a scalar confidence c_k, while tool embeddings are fixed throughout. At inference, CPHB allocates retrieval budget proportional to learned confidence; the resulting CPHB ranking is combined with a fused-embedding ranking via RRF to yield the final top-K result.

Confidence gates. Each head produces a scalar confidence from the backbone output, independent of its own projection:

$$c_k = \sigma\left(\mathbf{w}_k^\top \mathbf{h}\right) \in [0, 1] \tag{3}$$

Reading c_k from $\mathbf{h}$, rather than from $\hat{\mathbf{q}}_k$, allows confidence to be estimated from the complete query representation before demand-specific projection.

Tool side. Tool embeddings are the raw frozen ST vectors $\mathbf{e}_s \in \mathbb{R}^{384}$, L2-normalised before scoring. There are no learned parameters on the tool side.

3.3 Training with Hungarian Matching

LatDec treats the correspondence between retrieval heads and target tools as a latent assignment problem: the training labels identify the required tools but do not specify which head should represent each one. For each training example, Hungarian matching dynamically pairs target tools with heads based on their current matching costs.

For each query in a batch, the cost matrix $\mathbf{C} \in \mathbb{R}^{H \times |T^*|}$ has entries $C_{kt} = 1 - \cos(\hat{\mathbf{q}}_k, \mathbf{e}_s^{(t)})$; minimising total cost maximises cosine alignment between heads and assigned targets. The Hungarian algorithm Kuhn (1955) finds a minimum-cost bipartite matching Carion et al. (2020) that assigns each target tool to a distinct head, ensuring a one-to-one head-to-target assignment for the matched pairs.

Assigned heads receive a confidence target of one and unassigned heads receive a target of zero. When $|T^*| \leq H$, every target is matched. When $|T^*| > H$, the solver returns exactly H matched pairs, leaving $|T^*| - H$ targets unmatched and excluded from the loss.

The training objective combines three terms:

$$\mathcal{L} = \mathcal{L}_{\text{Hungarian}} + \lambda_{\text{InfoNCE}}\,\mathcal{L}_{\text{InfoNCE}} + \lambda_{\text{sparse}}\,\mathcal{L}_{\text{sparse}} \tag{4}$$

$\mathcal{L}_{\text{Hungarian}}$ combines cosine distance between each head and its assigned target tool with binary cross-entropy (BCE) supervision on confidence gates (assigned heads toward one, unassigned toward zero); $\mathcal{L}_{\text{InfoNCE}}$ (van den Oord et al., 2018) pushes each matched head toward its target tool and away from hard same-server negatives, drawing on established hard-negative training strategies for dense retrieval (Xiong et al., 2021; Qu et al., 2021); and $\mathcal{L}_{\text{sparse}}$ regularises inactive head confidences toward zero to prevent dormant heads from consuming retrieval budget. Full derivation of each term is in Appendix E.

3.4 Retrieval Mechanism

At inference, each head k scores all N tools by $\cos(\hat{\mathbf{q}}_k, \mathbf{e}_s)$. Producing a single top-K result from H ranked lists involves two components: Confidence-Proportional Head Budgeting (CPHB), which allocates retrieval budget across heads in proportion to their learned confidence, and embedding-space fusion, which blends the learned head representations with the frozen base query embedding to produce a complementary ranking anchored by that embedding. These two components combine via rank aggregation to create the final top-K ranked list of tools.

Confidence-Proportional Head Budgeting. Without a budget constraint, taking the global top-K across all heads lets the highest-scoring head dominate the result, leaving secondary demands under-represented on multi-tool queries. **Confidence-Proportional Head Budgeting (CPHB)** instead allocates each head a budget proportional to its learned confidence c_k before any cross-head comparison, then processes heads in descending confidence order, allowing each head to select its allocated number of previously unselected tools (full pseudocode in Appendix H); because c_k is trained to distinguish assigned heads from inactive heads, the allocation uses a learned estimate of head activity while cross-head deduplication allows active heads to contribute distinct tools within the total budget.

Embedding-space fusion. Alongside CPHB, we form a complementary query representation by combining all H head projections using their normalized confidence scores as weights:

$$\mathbf{q}_{\text{pop}} = \text{normalize}\left(\sum_{k=1}^{H} \frac{c_k}{\sum_{k'} c_{k'}}\,\hat{\mathbf{q}}_k\right) \in \mathbb{R}^{384} \tag{5}$$

We further blend it with $\mathbf{e}_q$, the frozen ST embedding of the query, which is independent of LatDec's task-specific training and provides a stable query-level reference:

$$\mathbf{q}_{\text{fused}} = \text{normalize}\left(\frac{\mathbf{q}_{\text{pop}} + \mathbf{e}_q}{2}\right) \in \mathbb{R}^{384} \tag{6}$$

When $\mathbf{q}_{\text{pop}}$ and $\mathbf{e}_q$ are closely aligned, their average remains close to both representations. When they diverge, the base query embedding provides a complementary query-level signal.

Final retrieval via Reciprocal Rank Fusion. We combine the ranked list from CPHB and the ranked list obtained by scoring all tools against $\mathbf{q}_{\text{fused}}$ via Reciprocal Rank Fusion (RRF) (Cormack et al., 2009), a parameter-light rank aggregation method that avoids introducing an additional learned or tuned weight between the two signals:

$$\text{score}(s) = \frac{1}{\kappa + \text{rank}_{\text{CPHB}}(s)} + \frac{1}{\kappa + \text{rank}_{\text{fused}}(s)} \tag{7}$$

where $\text{rank}_{\text{CPHB}}(s)$ and $\text{rank}_{\text{fused}}(s)$ are tool s's position in the CPHB ranking and the fused-embedding ranking respectively, and $\kappa = 60$ is the standard smoothing constant from Cormack et al. (2009), fixed rather than tuned. The final retrieved set is the top-K tools by $\text{score}(s)$. This is the retrieval mechanism used throughout the remainder of the paper; we report CPHB alone and embedding fusion alone as ablations in Section 5.3 to isolate each component's contribution.

4 CapCorpus Dataset

CapCorpus is a 22,602-tool MCP retrieval benchmark constructed to provide controlled coverage across query-complexity levels. Tools are collected from a public MCP registry (Smithery, 2024), comprising 18,907 unique tool names across 2,101 servers; queries are LLM-synthesized and grounded in these tool descriptions. Target tool labels are assigned by embedding similarity within each corresponding server's tool set; this automated assignment underwent human validation by four annotators. The study contains 169 uniquely assigned items, including 139 true-label items and 30 hard substitutions, plus a separate 50-item overlap set seen by all annotators. The true-label acceptance rate is 86.3%, and agreement on the overlap set is Fleiss' κ=0.342 (full protocol in Appendix D). The resulting native dataset is heavily skewed toward lower $|T^*|$ (Table 1). We first partition native plan identifiers into fixed train, validation, and test manifests using seed 42. Only after those manifests are fixed do we create combinatorial augmentation, by concatenating disjoint native training queries to populate the high-$|T^*|$ tail. Thus, neither validation nor test queries, nor any of their source plans, can enter the augmented training data. The final training set contains 68,289 native and 60,000 augmented queries (128,289 total); validation and test contain 14,633 and 14,634 native queries, respectively. Both evaluation partitions are native-only. Model checkpoints are selected using native validation data, and the test manifest remains locked until head-count selection is complete. Appendix C gives the complete dataset-construction and split-integrity protocol.

Zero-shot evaluation. We additionally evaluate zero-shot out-of-distribution generalization on LiveMCPBench Mo et al. (2025), an independent benchmark of N=95 human-authored tasks over a disjoint tool corpus.

Table 1: $|T^*|$ distribution of generated data before and after combinatorial augmentation. These counts describe the full generated pool before experimental manifest filtering, not the final experimental partitions. Augmentation adds 30,000 synthetic combinations at each of $|T^*| \in \{4, 5\}$ to ensure coverage at high demand complexity, and all augmented data used in the experiments are assigned exclusively to training (Appendix C).

	Native	**Native + augmented**
Total queries	97,570	157,570
Mean $\lvert T^*\rvert$	2.20	3.08
$\lvert T^*\rvert = 2$	78,838 (80.8%)	78,838 (50.0%)
$\lvert T^*\rvert = 3$	17,799 (18.2%)	17,799 (11.3%)
$\lvert T^*\rvert = 4$	930 (1.0%)	30,930 (19.6%)
$\lvert T^*\rvert = 5$	3 (0.0%)	30,003 (19.0%)

5 Experiments

5.1 Experimental Setup

Baselines. We compare LatDec against ST cosine (cosine similarity with frozen `all-MiniLM-L6-v2` Sentence Transformer embeddings Wang et al. (2020), no learned parameters), BiEncoder-128 and BiEncoder-384 (bi-encoders differing in projection dimensionality Reimers & Gurevych (2019); Karpukhin et al. (2020)), and a ColBERT-style multi-vector retriever Khattab & Zaharia (2020). All baselines are trained on the same CapCorpus training split as LatDec. Unless otherwise noted, LatDec refers to the full CPHB+Fusion retrieval mechanism (Section 3.4); we additionally report CPHB alone and embedding fusion alone as ablations.

Evaluation metric. All results are reported as Recall@10 at the tool level. 95% confidence intervals were computed using 10,000 paired percentile-bootstrap resamples with a fixed bootstrap seed of 42.

Table 2: CapCorpus overall Recall@10, test split (N=14,634). 95% CIs from 10,000 bootstrap resamples.

Method	Recall@10	95% CI
ST cosine	0.4009	[0.3958, 0.4060]
ColBERT	0.5087	[0.5041, 0.5134]
BiEncoder-128	0.5365	[0.5318, 0.5413]
BiEncoder-384	0.5327	[0.5279, 0.5375]
LatDec (CPHB+Fusion)	**0.5694**	[0.5646, 0.5744]

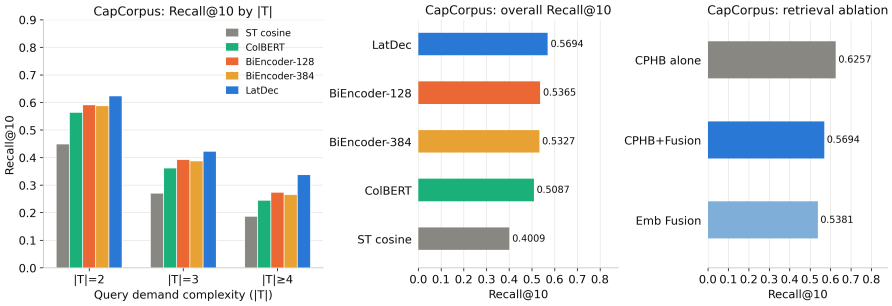

Figure 3: CapCorpus results (test split). Left: Recall@10 by query complexity $|T|$. Center: overall Recall@10 across methods. Right: retrieval mechanism ablation, isolating the contribution of CPHB and embedding fusion.

Retrieval head configuration. We select H=3 via a sweep over $H \in \{3, 5, 8, 10\}$ evaluated on the native validation partition: validation Recall@10 decreases monotonically from H=3 (0.589) through H=5 (0.559), H=8 (0.540), and H=10 (0.522), while the tool-subspace-relative redundancy score (derived from principal-angle overlap Hamm & Lee (2008); Hu et al. (2022)) rises from 0.236 at H=3 to 0.514 at H=8 and remains essentially unchanged at 0.513 for H=10. Together with decreasing per-head coverage, this indicates increasingly redundant rather than complementary demand representations (full sweep in Appendix G).

5.2 Tool Retrieval on CapCorpus

Table 2 and Figure 3 report overall and complexity-stratified Recall@10 on the CapCorpus test split. LatDec (CPHB+Fusion) outperforms every baseline on the held-out CapCorpus test split; all pairwise differences exclude zero at the 95% level (Appendix Table 3).

The complexity-stratified panel (Figure 3, left) shows this advantage is not uniform across query difficulty: LatDec's margin over the strongest baseline is +0.033 at $|T|$=2, +0.030 at $|T|$=3, and +0.064 at $|T|\geq 4$. The final value is the largest observed point-estimate margin, but the $|T|\geq 4$ test bucket contains only 156 queries and should be interpreted cautiously.

5.3 Retrieval Mechanism Ablation

Figure 3 (right) isolates the two components of the retrieval mechanism described in Section 3.4. With the selected H=3 model, CPHB alone achieves 0.6257 on the held-out CapCorpus test split, while CPHB+Fusion achieves 0.5694 (Δ=−0.056, 95% CI $[-0.060, -0.052]$): embedding-space fusion does not improve performance on the held-out CapCorpus test split at this head count. Embedding fusion alone achieves 0.5381 (Δ=+0.031 vs. CPHB+Fusion, 95% CI $[+0.027, +0.035]$). CPHB's learned routing is the primary driver of retrieval quality on the held-out CapCorpus test split; fusion's benefit is concentrated in out-of-distribution generalization, where it provides a substantial gain over CPHB alone (Section 5.4).

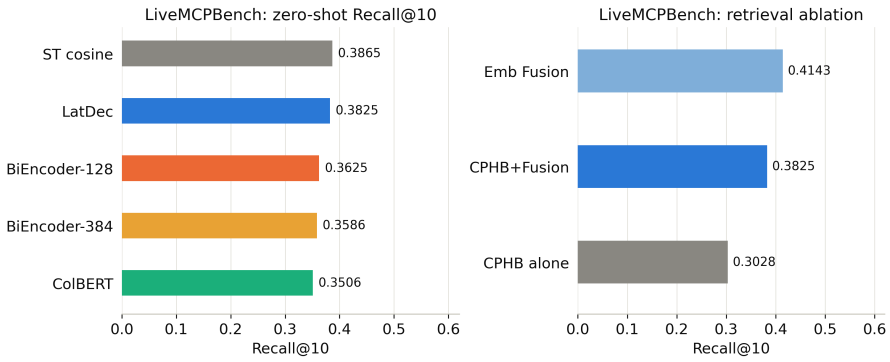

Figure 4: Zero-shot evaluation on LiveMCPBench (N=95). Left: Recall@10 for the full LatDec model (CPHB+Fusion) and the baselines; paired-bootstrap 95% confidence intervals for all differences between LatDec and the baselines include zero (Appendix Table 4). Right: Retrieval-mechanism ablation; adding embedding-space fusion to CPHB yields an improvement of +0.0797 (95% CI $[0.0445, 0.1172]$).

5.4 Zero-Shot Tool Retrieval on LiveMCPBench

We additionally evaluate LatDec zero-shot, without fine-tuning, on LiveMCPBench Mo et al. (2025), an independent benchmark of N=95 human-authored tasks over a disjoint tool corpus. Since both the queries and tools differ from CapCorpus, this evaluation serves as a boundary characterization rather than a direct comparison of performance across the two benchmarks. Given LiveMCPBench's modest size, we report all differences with bootstrap confidence intervals.

Baseline comparison. Figure 4 (left) summarizes the performance of LatDec and the baselines on LiveMCPBench. Paired bootstrap analysis shows that all paired-bootstrap confidence intervals for differences between LatDec and the baselines include zero (Table 4). For example, the difference versus ST cosine is Δ=−0.004 (95% CI $[-0.055, +0.046]$). The results therefore do not support a conclusive ranking of these methods at the current sample size.

Effect of embedding-space fusion. As shown in Figure 4 (right), CPHB alone achieves 0.3028, while adding embedding-space fusion increases performance to 0.3825, an improvement of +0.0797 (95% CI $[0.0445, 0.1172]$). Embedding fusion alone achieves the highest point estimate on LiveMCPBench (0.4143); however, its paired difference versus CPHB+Fusion is inconclusive (Δ=−0.032, 95% CI $[-0.071, +0.004]$).

Additional diagnostics show that fusion provides larger gains when the confidence-weighted population representation is less aligned with the frozen base query embedding (Appendix F). This suggests that effective decomposition requires a balance between head-level specialization and population-level semantic fidelity: individual heads should capture distinct demands, while their aggregate representation remains grounded in the semantics of the complete query.

6 Discussion

6.1 Representing Multiple Tool Demands

LatDec is motivated by the view that multi-tool retrieval requires representing the query's distinct demands separately, rather than scoring every candidate tool against a single compressed query representation. A single embedding may favor one demand while underrepresenting others, or it may occupy a compromise region that is not sufficiently aligned with the required tools. As shown in Figure 3 (left), LatDec's largest observed point-estimate margin occurs for queries requiring at least four tools, consistent with the value of representing multiple capability demands separately, although this 156-query bucket is small and the

pattern should be interpreted cautiously. Unlike external LLM decomposition, which may require access to tool metadata to generate capability-aware subqueries and consequently increase context and inference costs, LatDec learns the decomposition as part of retrieval training and produces multiple latent demand representations in a single encoder forward pass, without an auxiliary generative call.

6.2 Balancing Specialization and Semantic Fidelity

Learning multiple representations introduces a balance between head-level specialization and population-level semantic fidelity. Individual heads must represent distinct demands, while their aggregate representation remains grounded in the meaning of the complete query. Because confidence-based allocation is learned from the training data, the relative influence of the heads may become less reliable when query patterns change. The learned heads provide demand-specific representations, whereas the frozen base query embedding provides a global representation that is independent of LatDec's task-specific training. Fusion combines these complementary signals, preserving the benefits of specialization while reducing dependence on learned confidence estimates when the retrieval setting changes.

6.3 Implications for Evolving Tool Catalogs

LatDec's asymmetric design separates tool indexing from query-side learning. Since tools remain in the frozen ST representation space, new tools can be encoded and added to the index without retraining tool-side parameters. However, this does not guarantee that the learned query decomposition will retrieve them effectively, since the heads and confidence estimates may reflect capability combinations and language patterns encountered during training. Evolving tool catalogs may therefore require monitoring or recalibration when their query patterns or capability composition change substantially.

6.4 Limitations

CapCorpus uses LLM-generated queries, which may not fully capture the linguistic diversity, ambiguity, and variability of naturally occurring user requests. Independent human validation on a sampled subset provides supporting evidence for alignment between the queries and assigned tools, while evaluation on a larger collection of human-authored queries would further strengthen the findings. In addition, part of the training data is produced through combinatorial augmentation. Although augmented queries are excluded from both validation and test evaluation, they may still influence the representations learned by the model. Finally, while LiveMCPBench provides an independent zero-shot evaluation, future benchmarks that vary tool and query distributions separately would allow these two forms of generalization to be evaluated independently.

7 Conclusion

We introduced LatDec, a retrieval architecture that represents multiple latent demands from a query in a single encoder forward pass. On the held-out CapCorpus test split, LatDec outperforms matched retrieval baselines. On LiveMCPBench, the ablation analysis shows that embedding-space fusion improves CPHB, while the benchmark's current sample size does not support a clear ranking of the full LatDec model against the baselines. Together, these findings highlight the importance of balancing demand-specific specialization with preservation of the query's overall semantics when retrieving tools for complex agentic tasks.

Reproducibility Statement

The model architecture, training configuration, evaluation procedure, and dataset-construction methodology are documented in the paper and appendices. No code, dataset, or checkpoint artifacts are distributed with this submission. Because the underlying third-party

registry may change over time and the original tool-description snapshot is not redistributed, exact reconstruction of the experimental corpus is not guaranteed.

Ethics Statement

Human validation study. The human validation study described in Appendix D involved four annotators who voluntarily reviewed a stratified sample of automatically generated query-tool label assignments. Annotators were informed of the study's purpose and were not shown which items were deliberately incorrect (trap) items, to preserve the validity of the discrimination check; no other information was withheld, and the task involved no sensitive, personal, or potentially distressing content. No personally identifiable information about annotators is included in any released artifact; individual judgments are reported only in aggregate, and annotator identity is not linked to specific responses in any public release.

Data provenance. The MCP tool corpus underlying CapCorpus is constructed from metadata consisting of server names, tool names, and tool descriptions obtained through a public registry API (Smithery, 2024). We used documented endpoints that did not require authentication at the time of collection, together with sequential requests, pagination, and backoff in response to rate limits. This metadata reflects tool definitions published by independent third-party MCP server developers and indexed by Smithery. We contacted Smithery directly to clarify the applicable research-use and redistribution terms prior to release. In the absence of an official response, we adopt a conservative release policy: we publish our generated queries, assigned labels, split information, evaluation materials, and collection methodology, but do not redistribute the underlying tool descriptions or schemas. No user-level or personal data is collected, processed, or released as part of this work.

Intended use and risks. LatDec is a retrieval architecture intended to improve tool selection in agentic AI systems. We are not aware of a specific dual-use or harm pathway distinct from the general risks already associated with agentic tool-use systems; improved retrieval accuracy for legitimate tool sets does not meaningfully expand an agent's action space beyond what the underlying tool catalog already permits.

AI Use Statement

Generative AI tools were used to assist with implementing portions of the code under direct author supervision and to generate the synthetic queries used in CapCorpus, as described in Appendix C. They were also used for related-work discovery and for drafting and language editing portions of the manuscript. The research problem, architecture, methodological design, experimental decisions, analysis, and interpretation of the results were determined by the author. All AI-assisted code, generated data, references, and manuscript text were reviewed and verified by the author. The author takes responsibility for the final content of this work.

References

Anthropic. Model context protocol. `https://modelcontextprotocol.io/`, 2024. Accessed: 2026-09-14.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, volume 12346 of *Lecture Notes in Computer Science*, pp. 213–229, Cham, Switzerland, 2020. Springer. doi: 10.1007/978-3-030-58452-8_13. URL `https://doi.org/10.1007/978-3-030-58452-8_13`.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. Re-Invoke: Tool invocation

rewriting for zero-shot tool retrieval. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4705–4726, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.270. URL https://aclanthology.org/2024.findings-emnlp.270/.

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. SkillRet: A large-scale benchmark for skill retrieval in LLM agents, 2026. URL https://arxiv.org/abs/2605.05726.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. 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*, SIGIR '09, pp. 758–759, Boston, Massachusetts, USA, 2009. Association for Computing Machinery. doi: 10.1145/1571941.1572114. URL https://doi.org/10.1145/1571941.1572114.

Yu Du, Fangyun Wei, and Hongyang Zhang. AnyTool: 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*, pp. 11812–11829, Vienna, Austria, July 2024. PMLR. URL https://proceedings.mlr.press/v235/du24h.html.

Shiqing Fan, Xichen Ding, Liang Zhang, and Linjian Mo. MCPToolBench++: A large scale AI agent model context protocol MCP tool use benchmark, 2025. URL https://arxiv.org/abs/2508.07575.

Xiang Fei, Xiawu Zheng, and Hao Feng. MCP-Zero: Active tool discovery for autonomous LLM agents, 2025. URL https://arxiv.org/abs/2506.01056.

Tiantian Gan and Qiyao Sun. RAG-MCP: Mitigating prompt bloat in LLM tool selection via retrieval-augmented generation, 2025. URL https://arxiv.org/abs/2505.03275.

Jihun Hamm and Daniel D. Lee. Grassmann discriminant analysis: A unifying view on subspace-based learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 376–383, Helsinki, Finland, 2008. Association for Computing Machinery. doi: 10.1145/1390156.1390204. URL https://doi.org/10.1145/1390156.1390204.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*, Virtual, April 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *The Eighth International Conference on Learning Representations*, Addis Ababa, Ethiopia, April 2020. URL https://openreview.net/forum?id=SkxgnnNFvH.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (eds.), *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, pp. 39–48, Virtual Event, China, July 2020. Association for Computing Machinery. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, March 1955. doi: 10.1002/nav.3800020109. URL https://doi.org/10.1002/nav.3800020109.

Feng Li, Hao Zhang, Shilong Liu, Jian Guo, Lionel M. Ni, and Lei Zhang. DN-DETR: Accelerate DETR training by introducing query DeNoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13619–13627, New Orleans, Louisiana, USA, June 2022. IEEE. doi: 10.1109/CVPR52688.2022.01325. URL https://openaccess.thecvf.com/content/CVPR2022/html/Li_DN-DETR_Accelerate_DETR_Training_by_Introducing_Query_DeNoising_CVPR_2022_paper.html.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021. doi: 10.1162/tacl_a_00369. URL https://aclanthology.org/2021.tacl-1.20/.

Elias Lumer, Pradeep Honaganahalli Basavaraju, Myles Mason, James A. Burke, and Vamse Kumar Subbiah. Graph RAG-tool fusion, 2025a. URL https://arxiv.org/abs/2502.07223.

Elias Lumer, Anmol Gulati, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A. Burke. ScaleMCP: Dynamic and auto-synchronizing model context protocol tools for LLM agents, 2025b. URL https://arxiv.org/abs/2505.06416.

Elias Lumer, Faheem Nizar, Anmol Gulati, Pradeep Honaganahalli Basavaraju, and Vamse Kumar Subbiah. Tool-to-agent retrieval: Bridging tools and agents for scalable LLM multi-agent systems, 2025c. URL https://arxiv.org/abs/2511.01854.

Elias Lumer, Vamse Kumar Subbiah, James A. Burke, Pradeep Honaganahalli Basavaraju, and Austin Huber. Toolshed: Scale tool-equipped agents with advanced RAG-tool fusion and tool knowledge bases. In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025)*, volume 3, pp. 1180–1191, Porto, Portugal, 2025d. SCITEPRESS – Science and Technology Publications. doi: 10.5220/0013303000003890. URL https://www.scitepress.org/Link.aspx?doi=10.5220/0013303000003890.

Guozhao Mo, Wenliang Zhong, Jiawei Chen, Qianhao Yuan, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. LiveMCPBench: Can agents navigate an ocean of MCP tools?, 2025. URL https://arxiv.org/abs/2508.01780.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565, Vancouver, BC, Canada, December 2024. Curran Associates, Inc. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

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. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, Vienna, Austria, May 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 5835–5847, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.466. URL https://aclanthology.org/2021.naacl-main.466/.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24497–24524, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1258. URL https://aclanthology.org/2025.findings-acl.1258/.

Smithery. Smithery MCP registry, 2024. URL https://smithery.ai/.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2018. URL https://arxiv.org/abs/1807.03748.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. ToolGen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations*, Singapore, April 2025. URL https://openreview.net/forum?id=XLMAMmowdY.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Zifei Wang, Wei Wen, Qiang Ji, Ruizhi Qiao, and Xing Sun. Skill is not document: A query-conditional benchmark and two-stage retriever for LLM agent skill routing, 2026. URL https://arxiv.org/abs/2606.03565.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *The Ninth International Conference on Learning Representations*, Virtual, May 2021. URL https://openreview.net/forum?id=zeFrfgyZln.

Shunyu Zhang, Yaobo Liang, Ming Gong, Daxin Jiang, and Nan Duan. Multi-view document representation learning for open-domain dense retrieval. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5990–6000, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.414. URL https://aclanthology.org/2022.acl-long.414/.

A Bootstrap Confidence-Interval Analysis

Tables 3 and 4 report paired bootstrap confidence-interval comparisons; Table 5 additionally reports the underlying per-method point estimates and confidence intervals on LiveMCP-Bench referenced in Section 5.4.

Table 3: Paired bootstrap differences on the CapCorpus native-only test split ($n = 14{,}634$), Recall@10, H=3 model. All pairwise differences exclude zero. CPHB alone outperforms CPHB+Fusion on the held-out CapCorpus test split; the pre-specified primary method (CPHB+Fusion) outperforms all baselines.

Model A	Model B	Diff (A − B)	95% CI
LatDec (CPHB+Fusion)	ST cosine	+0.169	[+0.164, +0.173]
LatDec (CPHB+Fusion)	ColBERT	+0.061	[+0.056, +0.065]
LatDec (CPHB+Fusion)	BiEncoder-128	+0.033	[+0.029, +0.037]
LatDec (CPHB+Fusion)	BiEncoder-384	+0.037	[+0.032, +0.041]
Retrieval mechanism ablation			
CPHB+Fusion	CPHB alone	−0.056	[−0.060, −0.052]
CPHB+Fusion	Embedding Fusion alone	+0.031	[+0.027, +0.035]

Table 4: Paired bootstrap differences on LiveMCPBench ($n = 95$), Recall@10. Confidence intervals for cross-method differences against LatDec include zero at this sample size; the interval for the internal effect of embedding fusion over CPHB alone excludes zero.

Model A	Model B	Diff (A − B)	95% CI	Excl. 0?
LatDec (CPHB+Fusion)	ST cosine	−0.004	[−0.055, +0.046]	No
LatDec (CPHB+Fusion)	ColBERT	+0.032	[−0.022, +0.090]	No
LatDec (CPHB+Fusion)	BiEncoder-128	+0.020	[−0.034, +0.078]	No
LatDec (CPHB+Fusion)	BiEncoder-384	+0.024	[−0.029, +0.078]	No
Retrieval mechanism ablation				
CPHB+Fusion	CPHB alone	+0.080	[+0.045, +0.117]	**Yes**
CPHB+Fusion	Embedding Fusion alone	−0.032	[−0.071, +0.004]	No
Emb. Fusion alone	ST cosine	+0.028	[−0.012, +0.070]	No

Table 5: LiveMCPBench zero-shot Recall@10 (N=95). 95% CIs from 10,000 bootstrap resamples. Confidence intervals for pairwise differences between LatDec (CPHB+Fusion) and the baselines include zero; the embedding-fusion-alone point estimate is the highest on this benchmark.

Method	Recall@10	95% CI
LatDec CPHB alone	0.3028	[0.229, 0.384]
ColBERT	0.3506	[0.272, 0.434]
BiEncoder-384	0.3586	[0.281, 0.441]
BiEncoder-128	0.3625	[0.282, 0.447]
LatDec (CPHB+Fusion)	0.3825	[0.302, 0.466]
ST cosine	0.3865	[0.313, 0.466]
Emb. Fusion alone	**0.4143**	[0.333, 0.498]

B Training Configuration

All results in this paper use checkpoint `latdec_v11_H3_dp0p10_lr1p00_ls0p005_aug_mcp.pt`, trained to epoch 88 with the following configuration, verified against the training script (`training/train_mhsp.py`) and `model/loss.py`.

Architecture. $H = 3$ retrieval heads (the operating point selected on the native validation partition; Appendix G); head dropout $p = 0.1$; ReLU activations in head projections.

Optimizer and schedule. AdamW, learning rate 10^{-3}, weight decay 10^{-4}. Linear warmup over 5 epochs, then cosine decay to zero over the remaining 95 epochs (100 epochs total).

Batch size and hard negatives. Batch size 256 (training), 512 (validation); 8 hard same-server negatives per positive pair; InfoNCE temperature $\tau = 0.07$. Gradients are clipped to a global L2 norm of 1.0 after each backward pass.

Data partitions and model selection. Training uses 68,289 native queries plus 60,000 train-only augmented queries. Within each training run, the checkpoint is selected without loading the test partition. The final head count is selected by CPHB+Fusion Recall@10 on the 14,633-query native validation manifest. Only after this choice is recorded is evaluation on the 14,634-query native test manifest authorized.

Loss weights. $\lambda_{\text{pos}} = 1.0$, $\lambda_{\text{conf}} = 0.5$, $\lambda_{\text{InfoNCE}} = 1.0$, $\lambda_{\text{sparse}} = 0.005$. λ_{InfoNCE} and λ_{sparse} are encoded in the checkpoint name (`lr1p00`, `ls0p005`); λ_{pos} and λ_{conf} are the defaults in `model/loss.py` (lines 25–26), not overridden at the command line.

C CapCorpus Construction Methodology

C.1 Tool Corpus

We collect 22,602 tools from the public MCP catalogue (Smithery, 2024), corresponding to 18,907 unique tool names across 2,101 distinct servers (the difference reflects tool names that recur across multiple servers). Each tool is a (server, name, description) triple. To reach servers beyond Smithery's popularity-sorted top-500 result cap, we issue 80 broad search queries spanning infrastructure, developer tools, communication, web, data, and security, deduplicating results by server identifier. Servers are retained only if they carry an English description of at least 20 words and expose at least two tools with non-trivial descriptions; up to 20 tools per server are kept, ranked by description length. Tool descriptions are embedded with a frozen `all-MiniLM-L6-v2` Sentence Transformer (ST) to form the tool embedding matrix $\mathbf{E} \in \mathbb{R}^{N \times 384}$, which is fixed throughout training and evaluation.

C.2 Query Synthesis

The demand side of CapCorpus is fully LLM-synthesized, grounded in real tool descriptions from the corpus above. Real multi-tool MCP usage logs at the scale required for training do not yet exist; LLM synthesis under controlled sampling allows precise coverage of server combinations and $|T^*|$ distribution targets that opportunistic log-mining cannot guarantee.

For each query, we sample an anchor MCP server and 1–4 additional servers by cosine similarity between server centroid embeddings. To encourage topical diversity during sampling, servers are grouped into five keyword-seeded diversity pools (spanning monitoring and operations, developer tools, data and databases, web and search, and communication); approximately 30% of queries deliberately sample the additional servers from outside the anchor's pool. These pools are a generation-time sampling control only: pool assignment is not exposed in the released dataset, is not used as a training or evaluation feature, and is not intended as a validated content taxonomy. A language model is prompted with the sampled server set to produce a realistic user query together with one sub-query per server describing its specific task. The prompt explicitly discourages generic operations language unless the sampled servers are genuinely monitoring tools, to reduce surface-level pool leakage into the query text.

Of 100,000 plan generation attempts, 97,570 succeeded (97.6%); the remaining 2,430 failures are LLM request timeouts exhausting a fixed retry budget, distributed throughout the attempt sequence rather than concentrated at any point, consistent with transient failures rather than a systematic issue.

Queries were generated using `gpt-5.4-nano` with temperature 0.9 and a maximum of 800 completion tokens; no generation seed was specified. Tool sampling used a fixed random seed of 42. The complete system and user prompts are included in the supplementary code (`data/mcp/generate_mcp_queries.py`).

C.3 Label Assignment

Target tool labels are assigned by embedding each LLM-generated sub-query with ST and selecting the highest-cosine-similarity tool within the corresponding server's tool set:

$$t_k^* \;=\; \arg\max_{t \in \mathcal{T}_k} \cos\big(\mathrm{ST}(s_k),\ \mathbf{e}_t\big), \tag{8}$$

where s_k is the LLM-generated sub-query for server k, $\mathcal{T}_k$ is its tool set, and $\mathbf{e}_t \in \mathbf{E}$ is the precomputed embedding for tool t. The frozen ST encoder places each sub-query and its candidate tools in a common embedding space for label assignment. Because each sub-query is explicitly authored for a specific server, the argmax selection is restricted to that server's tool set rather than performed over the full corpus. We assess the resulting assignments against independent human judgment in Appendix D.

This procedure yields 97,570 base (query, T^*) pairs; the $|T^*|$ distribution is skewed: $|T^*| = 2$ accounts for 80.8% of queries (mean $|T^*| = 2.20$; Table 1).

C.4 Combinatorial Augmentation

A model trained only on the base distribution would have little exposure to high-demand queries, which are precisely the cases where single-vector retrieval fails most severely: native queries have only 930 examples at $|T^*|$=4 and 3 at $|T^*|$=5. For each target $|T^*| \in \{4, 5\}$, we sample sets of base queries with disjoint skill sets (drawn from different skill-index buckets), concatenate their query texts using rule-based connectors (e.g. *"{A}. Meanwhile, {B}. Also, {C}."*), re-embed the combined text with ST, and union their T^* sets, producing 30,000 synthetic combinations at each target level. At $|T^*|$=3, the disjoint-skill-set sampling constraint was not satisfiable within the generation procedure's early-exit threshold, so no augmented rows were produced at this level; the $|T^*|$=3 bucket consists entirely of the 17,799 native examples. The full generated pool therefore contains 97,570 native and 60,000 template-concatenated records (Table 1). The table reports counts before experimental manifest filtering; the experimental partitions are defined separately as follows.

Split-integrity and augmentation protocol. The experimental split is defined on native plan identifiers *before* augmentation. The three native manifests are disjoint and fixed using seed 42. The final manifests contain 68,289 native training queries, 14,633 native validation queries, and 14,634 native test queries. These manifests contain 97,556 native records in total; 14 records from the generated native dataset are outside the experimental manifest set.

We construct all 60,000 augmented queries exclusively from source plans in the native training manifest, with 30,000 examples at each of $|T^*| \in \{4, 5\}$. Every augmented query is assigned to training. No augmented query is placed in validation or test, and no native validation or test plan can serve as a source for an augmented training query. The final training set therefore has 128,289 examples (68,289 native plus 60,000 augmented), while validation and test remain native-only.

Training and within-run checkpoint selection do not load the test manifest. The number of heads is selected by CPHB+Fusion Recall@10 on the native validation manifest. A separate selection record keeps the test partition locked until this choice is complete; the canonical test evaluation is run only after that record authorizes access. The train, validation, and test manifest SHA-256 hashes are, respectively, `f9d463e9d79a47569c128813b25404af3aeecb352aa33151bebabcd957aba91e`, `ff638ec2afbe56c449163d71c22a29d53ccd479926a83e20e8580c2f88770927` and `4bbf2e8516d7ec944acc2ae3119ee00d6a93e2ae6ef605a16bb694ca80a4a652`. All Cap-Corpus results in Section 5 use the latter 14,634-query native test manifest.

D Human Validation Protocol

We validate the argmax label assignment procedure (Appendix C) against independent human judgment by four annotators. The study contains 169 uniquely assigned (query, tool) items drawn across $|T^*|$ levels and domain pools, plus a separate set of 50 overlap items seen by all four annotators. Annotators were shown the full query and one candidate tool (with its same-server siblings visible as context) and asked whether the candidate is a correct or reasonable match for some part of the query; annotators were explicitly instructed that a query may require multiple tools and that a partial match counts as a valid match, so a positive judgment does not assert that the shown tool is the *unique* correct answer, only that it is a genuine one.

Human acceptance rate. On 139 items where the displayed candidate was the true argmax-assigned tool, annotators judged 120 as correct, a human acceptance rate of 86.3%. We use "acceptance rate" rather than "precision" to avoid conflating this quantity with the standard information-retrieval definition of precision, which assumes an independent ground truth; here, human judgment is being compared against the same automated procedure being validated, not an external gold standard.

Discrimination against hard negatives. Endorsement alone cannot distinguish genuine label verification from passive agreement with whatever is presented. To probe this, a disjoint set of 30 items instead displayed a deliberately incorrect tool in place of the true assignment: specifically, the same-server sibling with the highest cosine similarity to the true tool, the hardest substitution available, with the true tool still shown among the visible alternatives. Annotators correctly rejected 16 of these 30 hard substitutions (53.3%), and among those 16 rejections, the tool they proposed as correct matched the true argmax assignment exactly in 12 cases (75.0%), indicating that successful rejections substantively converge on the correct tool rather than reflecting arbitrary disagreement. Because this discrimination check uses maximally confusable pairs by construction, we do not treat the acceptance rate and the substitution rejection rate as directly comparable quantities, nor do we interpret their difference as evidence of a specific bias; the two conditions test different things (endorsement of a true label versus rejection of the hardest available false one) and are not linked by a within-item crossover design that would support such a comparison. We report both because each addresses a distinct failure mode a single acceptance rate cannot rule out on its own.

Inter-annotator reliability. On a separate set of 50 items annotated by all four annotators (200 judgments total), we measure inter-annotator reliability independent of agreement with the automated labels: 32 of 50 items (64.0%) receive a unanimous judgment across all four annotators, and Fleiss' $\kappa = 0.342$.

Taken together, these results provide supporting evidence that the sampled argmax-assigned labels are often judged reasonable by independent annotators and that annotators do not simply endorse every presented candidate. The remaining disagreement also illustrates the ambiguity involved in assigning individual tools to multi-demand queries.

E Training Objective

Notation. Let B be the mini-batch size and H the number of retrieval heads. For example $b \in \{1, \ldots, B\}$, let T_b^* be its target-tool set, $\hat{\mathbf{q}}_{b,k} \in \mathbb{R}^{384}$ the L2-normalised projection of head $k \in \{1, \ldots, H\}$, $\mathbf{e}_t \in \mathbb{R}^{384}$ the frozen L2-normalised Sentence Transformer embedding of tool t, and $c_{b,k} \in (0, 1)$ the post-sigmoid confidence of head k. Superscript $*$ denotes ground truth; a hat denotes a learned, normalised query projection.

Hungarian assignment. For each example, we construct a cost matrix $\mathbf{C}_b \in \mathbb{R}^{H \times |T_b^*|}$ with entries

$$C_b[k, t] = 1 - \hat{\mathbf{q}}_{b,k}^\top \mathbf{e}_t, \qquad k \in \{1, \ldots, H\},\ t \in T_b^*. \tag{9}$$

Solving the linear-sum assignment problem yields a set M_b of matched head-target pairs. If $|T_b^*| \leq H$, all targets are matched. If $|T_b^*| > H$, the solver returns exactly H pairs and the remaining $|T_b^*| - H$ targets do not contribute to the loss. No target is selected in advance; the joint minimum-cost assignment determines which H targets are matched. With the selected H=3, this second case occurs for training examples with $|T^*| \in \{4, 5\}$.

Let $A_b = \{k : \exists t \text{ such that } (k,t) \in M_b\}$ be the assigned heads for example b, and let $M = \{(b,k,t) : (k,t) \in M_b\}$ collect the matched triples across the batch. Consequently, $|M| = \sum_{b=1}^{B} \min(H, |T_b^*|)$. Define the confidence target $y_{b,k} = 1$ when $k \in A_b$ and $y_{b,k} = 0$ otherwise.

Positive alignment loss.

$$\mathcal{L}_{\text{pos}} = \frac{1}{|M|} \sum_{(b,k,t)\in M} \left(1 - \hat{\mathbf{q}}_{b,k}^{\top} \mathbf{e}_t\right) \tag{10}$$

is a flat mean over every matched pair in the mini-batch; because $|T_b^*|$ varies across examples, a query contributes one term per matched target, up to H terms.

Confidence loss.

$$\mathcal{L}_{\text{conf}} = \frac{1}{BH} \sum_{b=1}^{B} \sum_{k=1}^{H} \text{BCE}(c_{b,k},\ y_{b,k}) \tag{11}$$

where $\text{BCE}(p, y) = -y \log p - (1-y) \log(1-p)$. The loss is applied to the post-sigmoid probability $c_{b,k}$ and is averaged as a flat mean over all $B \times H$ entries.

Contrastive loss. For each matched triple $(b,k,t) \in M$, define two negative collections. $\mathcal{H}_{b,k,t}$ contains up to eight other tools from the same MCP server as t, sampled uniformly without replacement; all available tools are used if fewer than eight exist, and the sample is redrawn each batch. $\mathcal{B}_{b,k,t}$ contains the tool embeddings from every other matched triple in the mini-batch:

$$\mathcal{B}_{b,k,t} = \{\mathbf{e}_{t'} : (b',k',t') \in M,\ (b',k',t') \neq (b,k,t)\}. \tag{12}$$

The exclusion is by matched-triple index, so an identical tool associated with another triple remains an in-batch negative. The implementation concatenates $\mathcal{H}_{b,k,t}$ and $\mathcal{B}_{b,k,t}$ without deduplicating repeated embeddings; denote this concatenated collection by $\mathcal{N}_{b,k,t}$. Let $M' \subseteq M$ contain the triples for which $\mathcal{N}_{b,k,t}$ is nonempty. Then

$$\mathcal{L}_{\text{InfoNCE}} = -\frac{1}{|M'|} \sum_{(b,k,t)\in M'} \log \frac{\exp\left(\hat{\mathbf{q}}_{b,k}^{\top} \mathbf{e}_t / \tau\right)}{\exp\left(\hat{\mathbf{q}}_{b,k}^{\top} \mathbf{e}_t / \tau\right) + \sum_{\mathbf{e}^- \in \mathcal{N}_{b,k,t}} \exp\left(\hat{\mathbf{q}}_{b,k}^{\top} \mathbf{e}^- / \tau\right)} \tag{13}$$

with temperature τ=0.07. When M' is empty, this component is defined as zero.

Sparsity regulariser.

$$\mathcal{L}_{\text{sparse}} = \frac{1}{BH} \sum_{b=1}^{B} \sum_{k=1}^{H} c_{b,k} \tag{14}$$

penalises all head confidences, including those of assigned heads, thereby discouraging overall confidence inflation.

Combined objective. For consistency with Section 3.3, define

$$\mathcal{L}_{\text{Hungarian}} = \lambda_{\text{pos}} \mathcal{L}_{\text{pos}} + \lambda_{\text{conf}} \mathcal{L}_{\text{conf}}. \tag{15}$$

The loss used for backpropagation is

$$\mathcal{L} = \mathcal{L}_{\text{Hungarian}} + \lambda_{\text{InfoNCE}} \mathcal{L}_{\text{InfoNCE}} + \lambda_{\text{sparse}} \mathcal{L}_{\text{sparse}}, \tag{16}$$

where λ_{pos}=1.0, λ_{conf}=0.5, λ_{InfoNCE}=1.0, and λ_{sparse}=0.005. Thus every symbol in the combined loss corresponds directly to one of the components defined above.

Because tool embeddings are frozen throughout training, head-target assignment is shaped only by the query-side projections rather than by the joint evolution of both sides of the match, a dynamic that has been linked to training instability in related set-prediction settings where both sides are learned jointly Li et al. (2022).

F Additional Analysis of Confidence Allocation and Fusion

All analyses in this section use the selected H=3 checkpoint (`latdec_v11_H3_dp0p10_lr1p00_ls0p005_aug_mcp.pt`). CapCorpus statistics are computed on the native-only test partition (N=14,634); LiveMCPBench statistics are computed on all 95 queries.

Uniform-budget comparison. We compare CPHB with uniform budget allocation, which assigns equal retrieval slots to all heads without using learned confidence. With H=3, CPHB and uniform allocation achieve identical Recall@10 (0.3028), indicating that the learned confidence scores provide no allocation advantage over uniform allocation on LiveMCPBench at this head count.

Per-head confidence. Mean per-head confidence is higher on LiveMCPBench than on CapCorpus overall (0.774 versus 0.745, $\Delta = +0.029$). The change is uneven across heads: two heads increase by 0.055 and 0.048, while one head decreases by 0.017. These differences indicate that the model's confidence allocation shifts unevenly when applied to LiveMCP-Bench.

Cosine alignment and fusion gain. Cosine similarity between the confidence-weighted population vector $\mathbf{q}_{\text{pop}}$ and the frozen base query embedding $\mathbf{e}_q$ is higher on CapCorpus (mean 0.603, standard deviation 0.093) than on LiveMCPBench (mean 0.488, standard deviation 0.098). Using micro-averaged Recall@10, queries in the lowest-similarity quartile gain more from fusion (+0.048, from 0.129 to 0.177) than those in the highest-similarity quartile (+0.027, from 0.329 to 0.356). This pattern is consistent with fusion being most beneficial when the learned population representation is less closely aligned with the frozen base query embedding.

G Head Count Sweep

We measure tool-subspace-relative inter-head redundancy using a Grassmann-based coverage metric Hamm & Lee (2008); Hu et al. (2022). For each head k, we collect the confidence-gated projections $\{\hat{\mathbf{q}}_k^{(i)}\}$ over the native validation partition and extract the subspace $\mathbf{U}_k \in \mathbb{R}^{384\times d_k}$ as the top-d_k right singular vectors accounting for 95% of the total squared singular-value energy. Per-head coverage against the tool-embedding subspace $\mathbf{U}_{\text{skills}}$, extracted analogously from all tool embeddings, is:

$$\varphi_k = \frac{1}{d_{\text{skills}}} \left\| \mathbf{U}_{\text{skills}}^\top \mathbf{U}_k \right\|_F^2 = \frac{1}{d_{\text{skills}}} \sum_{i=1}^{\min(d_{\text{skills}},\, d_k)} \sigma_i^2 \tag{17}$$

where σ_i are the singular values of $\mathbf{U}_{\text{skills}}^\top \mathbf{U}_k$, equivalently the cosines of the principal angles between the subspaces. When $\sum_{k=1}^{H} \varphi_k > 0$, we define the tool-subspace-relative redundancy score ρ as:

$$\rho = \frac{\sum_{k=1}^{H} \varphi_k - \varphi_{\text{joint}}}{\sum_{k=1}^{H} \varphi_k} \tag{18}$$

where φ_{joint} is the coverage of the union subspace $\mathbf{U}_{\text{joint}} = \text{orth}([\mathbf{U}_1, \ldots, \mathbf{U}_H])$, with $\text{orth}(\cdot)$ denoting an orthonormal basis for the column span. We use this quantity as a diagnostic

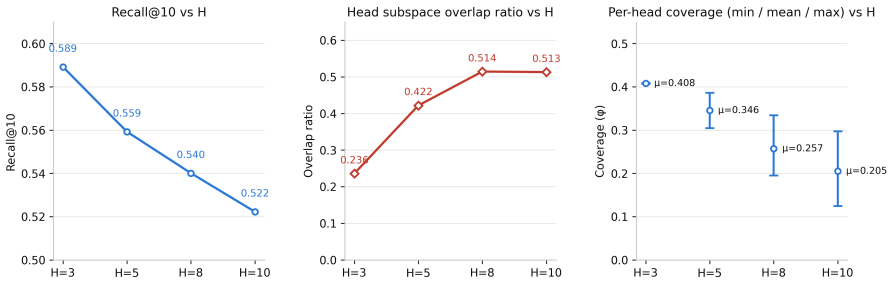

Figure 5: Effect of head count H on CapCorpus (native validation partition), using the CPHB+Fusion retrieval mechanism. Recall@10 decreases monotonically from H=3 to H=10; H=3 is selected as the paper model. The tool-subspace-relative redundancy score ρ rises through H=8 and remains near the same level at H=10 (centre), while per-head coverage decreases throughout (right).

of repeated coverage relative to the tool-embedding subspace. It is not guaranteed to be nonnegative in general, although all values observed in our head-count sweep are positive; within these observed values, higher scores indicate greater redundancy relative to joint coverage.

Figure 5 reports results on the CapCorpus native validation partition (N=14,633). Recall@10 decreases monotonically from H=3 (0.589) to H=5 (0.559), H=8 (0.540), and H=10 (0.522); H=3 achieves the highest validation Recall@10 and is selected as the paper model. The redundancy score ρ rises from 0.236 at H=3 to 0.422 at H=5 and 0.514 at H=8, then remains essentially unchanged at 0.513 for H=10. Mean per-head coverage decreases from 0.408 to 0.205. Together, these diagnostics indicate that additional heads produce increasingly redundant rather than complementary demand representations.

H CPHB Algorithm Detail

Require: Head projections $\{\hat{\mathbf{q}}_h\}_{h=1}^{H}$ (unit-normalised), confidences $\{c_h\}_{h=1}^{H}$, indexed tool embeddings $\{\mathbf{e}_s\}_{s=1}^{S}$, and total budget K.
Ensure: At most K unique tools, ranked by their recorded cosine scores.

1. Set $\tau \leftarrow \sum_{h=1}^{H} c_h$. If numerical underflow produces $\tau = 0$, return $\emptyset$.
2. For each head, set $b_h \leftarrow \lfloor (c_h/\tau)K \rfloor$.
3. Distribute the remaining $K - \sum_h b_h$ slots to heads with the largest fractional remainders.
4. Set $\pi \leftarrow \text{argsort}(\{c_h\})$ in descending order and seen $\leftarrow \{\}$.
5. For each $h \in \pi$ with $b_h > 0$:
   (a) Set added $\leftarrow 0$.
   (b) Traverse the head's candidate list in descending order of $\cos(\hat{\mathbf{q}}_h, \mathbf{e}_s)$. Skip tools already in seen; otherwise record this head's score and increment added. Stop when added $= b_h$ or the candidate list is exhausted.
6. Return the tools in seen ranked by their recorded scores.

Algorithm 1: Confidence-Proportional Head Budgeting (CPHB)

For computational efficiency, the implementation forms each head's candidate list from its top P=30 tools and uses K=10. Since heads are processed sequentially with a shared deduplication set, $P > K$ provides slack when their lists overlap, but it does not make the truncated procedure mathematically identical to full-corpus traversal. In a pathological case with extensive cross-head overlap, a head can exhaust its 30 candidates before filling its budget, so the algorithm may return fewer than K unique tools. The pseudocode states this behavior explicitly through the candidate-list exhaustion condition.