CoHyDE: Iterative Co-Training of LLM Rewriter & Dense Encoder for Tool Retrieval

Anonymous ACL submission

Abstract

Tool retrieval over large API catalogs is a core bottleneck for LLM agents: user queries arrive in colloquial, often underspecified language, while the catalog uses technical API vocabulary that no fixed encoder can bridge on its own. The two dominant training approaches, contrastive encoder fine-tuning and HyDE-style query expansion with a frozen LLM, address this problem from opposite ends and fail in complementary directions: the fine-tuned encoder excels when the query's surface form already matches the catalog but collapses when it does not, while zero-shot HyDE is more robust to underspecified queries yet generates catalog-unaware hypothetical descriptions that degrade retrieval when queries are well-formed. We introduce **CoHyDE**, an iterative procedure that trains the dense encoder and the LLM rewriter as a single co-evolving system: the encoder is retrained with InfoNCE on catalog-style hypothetical descriptions produced by the rewriter, and the rewriter is preference-aligned via DPO against the encoder's retrieval scores, with both sides warm-started on the tool catalog before the loop begins. On a $\sim$10k tool subset of the ToolBench catalog (Qin et al., 2024), three rounds of CoHyDE improve over the strongest single-component baseline by **+2.5 pp** NDCG@5 on standard queries and **+6.3 pp** on held-out vague queries, with gains as large as **+8 pp** on the hardest vague tier. Ablations confirm that co-training is the key ingredient: using either component in isolation fails to match CoHyDE on both well-formed and vague queries, with losses of up to **-8 pp** on vague queries.

1 Introduction

Modern language model agents act in the world by calling external tools drawn from catalogs that increasingly number in the tens of thousands (Qin et al., 2024; Patil et al., 2024). No agent can fit every tool's documentation into its context window, and the quality of an agent's actions is bounded above by an upstream *tool retrieval* step that selects a small candidate set per user query.

The dominant retrieval recipe embeds queries and tools into a shared vector space and returns the top-k most similar tools by nearest-neighbor lookup. Two largely disjoint research directions have grown around this recipe. **Direction 1: query expansion with a frozen LLM.** HyDE-style methods (Gao et al., 2023; Wang et al., 2023) prompt a frozen LLM to generate a hypothetical document for the query and search a frozen encoder against its embedding. **Direction 2: encoder fine-tuning with no query rewriting.** Dense-retrieval methods fine-tune the encoder on (query, tool) pairs with contrastive losses (Karpukhin et al., 2020; Xiao et al., 2024).

Both directions have a complementary failure mode. A trained dense encoder is, in essence, a similarity function shaped by the (anchor, positive) pairs it sees during training. When the query is in-distribution (i.e., sharing lexical surface with the catalog), the contrastive signal is sufficient; when surface form drifts, the encoder has no world-knowledge or reasoning machinery to bridge the gap and falls back on residual lexical cues (Thakur et al., 2021; Chen et al., 2022). Query-expansion approaches fail symmetrically: the LLM brings the reasoning needed to handle vague queries (Wei et al., 2022), but its generated output does not match the catalog's vocabulary, so on well-formed queries, it hurts more than it helps (Lei et al., 2024). This raises a natural question: *can the two training modes be combined into a single procedure that is stronger than either component alone?*

We introduce **CoHyDE**, an iterative co-training procedure that treats the dense encoder and the LLM rewriter as a single co-evolving system. In each round, the LLM generates catalog-style hypothetical descriptions for each query; the encoder is then retrained via contrastive learning on these

descriptions, and the LLM is preference-aligned via DPO using the encoder's own retrieval scores as reward signal. This alternating update cycle is repeated for multiple iterations, with each component progressively adapting to the other.

We apply CoHyDE on a ∼10k-tool subset of the ToolBench catalog (Qin et al., 2024). After three rounds of co-training, CoHyDE improves over the strongest single-component baseline by **+2.5 pp** NDCG@5 on standard queries and **+6.3 pp** on held-out vague queries, with gains as large as **+8 pp** on the hardest vague tier.

To summarize our contributions: (i) We introduce CoHyDE, an iterative co-training procedure that jointly optimizes a dense encoder and an LLM rewriter for tool retrieval. (ii) We empirically characterize the complementary failure modes of encoder fine-tuning and zero-shot HyDE, motivating the need to train both components jointly.

2 Related Work

Tool retrieval. Dense tool-retrieval methods fine-tune an encoder on (query, tool) pairs with contrastive supervision (Qin et al., 2024; Anantha et al., 2023; Qu et al., 2024; Shi et al., 2025); a parallel line treats retrieval as a frozen black-box via LLM-based expansion or generative indexing (Patil et al., 2024; Chen et al., 2024; Lumer et al., 2025; Wang et al., 2025). The closest prior work is Shao et al. (2023), which iteratively rewrites user instructions and retrains the encoder on (rewritten-instruction, tool) pairs. CoHyDE differs: the rewriter is preference-aligned via DPO against the encoder it feeds, rewrites target *catalog-description style* rather than query style, and the encoder retrain uses no real (query, tool) pairs. A concurrent line of work (Anonymous, 2026) audits *parametric* tool retrieval, where tools are embedded as virtual tokens in an LLM's vocabulary (Wang et al., 2025); this paradigm is orthogonal to CoHyDE, which improves dense encoder retrieval.

Query expansion and trained rewriters. HyDE (Gao et al., 2023) searches a frozen index against a hypothetical document embedding; Query2doc (Wang et al., 2023) concatenates the pseudo-document to the original query. CSQE (Lei et al., 2024) patches corpus-misalignment of LLM expansions at test time by injecting retrieved sentences; we address the same misalignment at training time. Trained query rewriters like Rewrite-Retrieve-Read (Ma et al., 2023), RaFe (Mao et al., 2024), and LeReT (Hsu et al., 2025) use RL or DPO with a *frozen* retriever; a complementary thread (Nogueira et al., 2019; Dai et al., 2023; Bonifacio et al., 2022; Wang et al., 2022) trains the retriever on LLM-generated synthetic queries with the generator frozen. All these methods freeze at least one component, whereas CoHyDE co-trains both.

Dense retriever robustness and joint retriever-generator training. Dense retrievers are brittle off-distribution (Thakur et al., 2021; Sciavolino et al., 2021; Chen et al., 2022; Yu et al., 2022); domain-adaptation via synthetic queries (Wang et al., 2022; Dai et al., 2023; Meng et al., 2024; Lin et al., 2023) runs the generation loop once with a frozen generator. Joint retriever–generator frameworks like RAG (Lewis et al., 2020), Atlas (Izacard et al., 2023), REPLUG (Shi et al., 2024), RA-DIT (Lin et al., 2024), Self-RAG (Asai et al., 2024) train the generator to produce better *final answers*, not better retrieval inputs. Prior work has therefore never co-trained a generator whose output *is* the retrieval input with the encoder that consumes it, the precise gap CoHyDE fills.

3 Methodology

3.1 Problem Formulation

Let $\mathcal{T} = \{t_1, \ldots, t_N\}$ denote a tool catalog of size N, where each tool $t \in \mathcal{T}$ carries a structured record (api name & description as well as tool title & description). We write $\phi : \mathcal{T} \to \Sigma^*$ for a fixed *rendering* function that serialises a tool into a single text string. Given a query $q \in \Sigma^*$ and a budget $k \in \mathbb{N}$, the tool-retrieval problem is to return a ranked set $\hat{T}_k(q) \subseteq \mathcal{T}$ with $|\hat{T}_k(q)| = k$ that maximally overlaps the gold tool set $T_q^* \subseteq \mathcal{T}$.

We restrict attention to single-vector dense encoder retrieval, the dominant architecture in tool retrieval (Qin et al., 2024; Anantha et al., 2023; Qu et al., 2024; Shi et al., 2025). A parameterised encoder $f_\theta : \Sigma^* \to \mathbb{R}^d$ maps any text into a d-dimensional unit-norm vector and retrieval is performed by approximate nearest-neighbour search (Johnson et al., 2021),

$$\hat{T}_k(q; \theta) = \mathrm{topk}_{t \in \mathcal{T}} \left\langle f_\theta(q),\ f_\theta\big(\phi(t)\big) \right\rangle \quad (1)$$

We additionally consider a *rewriter-augmented* variant in which a generator $g_\psi : \Sigma^* \to \Sigma^*$ produces a hypothetical tool description $\tilde{d} = g_\psi(q)$

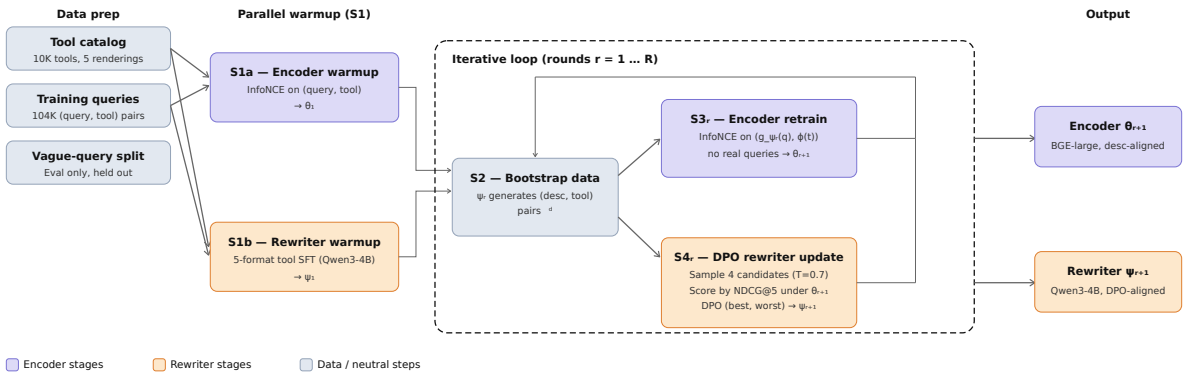

Figure 1: Overview of CoHyDE: a dense encoder and an LLM rewriter are co-trained in an alternating loop, with each component iteratively adapted to the other.

that is encoded *in place of* the query:

$$\hat{T}_k^{g_\psi}(q;\theta) = \mathrm{topk}_{t\in\mathcal{T}} \left\langle f_\theta\big(g_\psi(q)\big),\ f_\theta\big(\phi(t)\big)\right\rangle. \tag{2}$$

The goal of CoHyDE is to find parameters (θ^*, ψ^*) such that the two components reinforce each other, which we achieve through an alternating sequence of encoder and rewriter updates described in §3.5.

3.2 Data

Tool catalog. The ToolBench API pool (Qin et al., 2024) contains $|\mathcal{T}_{\text{full}}| = 46{,}980$ tools, partitioned into three evaluation tiers: single-domain (G1), cross-domain same-category (G2), and cross-domain different-category (G3), with 1,092 official evaluation queries (593 / 399 / 100 over G1/G2/G3). We work with a stratified subset $\mathcal{T}$ of $N = 10{,}000$ tools sized for training tractability: the subset retains every tool referenced by the gold sets of the evaluation queries, and stratified-samples the remaining slots to preserve the per-tier proportions of $\mathcal{T}_{\text{full}}$.

Training set. The training set $\mathcal{D}_{\text{train}} = \{(q_i, T^*_{q_i})\}_{i=1}^{M}$ consists of $M = 104{,}224$ (query, gold-tool-set) pairs (44,873 / 35,402 / 23,949 over G1/G2/G3); most queries have multiple gold tools ($|T^*_q| > 1$ for 93–99% of q). For contrastive training, we flatten these to individual (query, tool) pairs $\mathcal{D}_{\text{q}} = \{(q, \phi(t)) : (q, T^*_q) \in \mathcal{D}_{\text{train}},\ t \in T^*_q\}$.

Tool rendering. We represent each tool under a family of five rendering conventions $\Phi = \{\phi_1, \ldots, \phi_5\}$ spanning its natural information axes: ϕ_1 (title only), ϕ_2 (+API name), ϕ_3 (+tool description), ϕ_4 (title, API name, API description), and ϕ_5 (full record). At training time, $\phi \sim \mathrm{Unif}(\Phi)$ is sampled independently per (query, tool) pair, so each tool is seen under all five surface forms over an epoch. This format mixture encourages the encoder to learn representations invariant to catalog-side surface variation, including the longer multi-sentence ϕ_5 that most closely matches the rewriter's output style. At inference, the catalog is indexed under ϕ_5.

Vague-query split. We adopt the vague-query evaluation protocol of Chen et al. (2026) to probe robustness under query-side distribution shift. Each $q \in \mathcal{Q}_{\text{eval}}$ is paraphrased to replace surface tokens with conversational alternatives, while preserving the original gold tool set. We follow the protocol of Chen et al. (2026) exactly, substituting claude-4.5-opus for the GPT-4o paraphraser used in the original work. $\mathcal{Q}_{\text{vague}}$ does not enter any training procedure; two-pass validation (LLM self-check on every paraphrase plus an author spot-check on 50 samples) is described in Appendix A.

3.3 Encoder

f_θ is initialised from BGE-large-en-v1.5 (Xiao et al., 2024) (≈335M parameters, $d = 1024$). Given $x \in \Sigma^*$ we define $f_\theta(x) = h^\theta_{\text{CLS}}(x)/\|h^\theta_{\text{CLS}}(x)\|_2$, and the same encoder is applied to queries, tool renderings, and rewriter outputs (a *symmetric* bi-encoder). Training minimises the symmetric InfoNCE loss (van den Oord et al., 2019) with temperature $\tau = 0.05$ and in-batch negatives; full loss expression and optimisation hyperparameters are in Appendix F.

We define two contrastive training datasets, differing only in what serves as the anchor: $\mathcal{D}_{\text{q}}$ pairs user queries with tool renderings, while $\mathcal{D}_{\text{d}}^{(\psi)}$ pairs rewriter-generated hypothetical descriptions $g_\psi(q)$ with tool renderings. In both cases the tool side is

rendered under a rendering ϕ sampled uniformly from Φ.

3.4 Rewriter

g_ψ is Qwen3.5-4B (Yang et al., 2025), an instruction-tuned decoder-only transformer. We define a prompt operator ρ_{HyDE} that wraps a query with an instruction to enumerate the full tool description of tool capable of fulfilling the query's intent, in catalog-style description format (Appendix C). A deterministic cleaning operator $\text{clean}(\cdot)$ strips reasoning-trace blocks and conversational preambles before encoding (Appendix B). At inference, the rewriter produces $\tilde{d} = \text{clean}(g_\psi(\rho_{\text{HyDE}}(q)))$ and retrieval proceeds against $\tilde{d}$ alone, replacing the original query entirely.

3.5 CoHyDE: Iterative Co-training

We index encoder and rewriter checkpoints by training stage: θ_i, ψ_i are the parameters after stage i. θ_0 denotes BGE-large-en-v1.5 pretrained weights; ψ_0 denotes the opensource instruction-tuned Qwen3.5-4B. The pipeline has two parallel warmup steps (S1a & S1b) followed by a bootstrap data-generation step (S2) and an alternating training loop (S3, S4) that may be unrolled for any number of rounds $R \geq 1$. Figure 1 and Algorithm 1 summarise the procedure.

S1a: Encoder warmup. The encoder is trained with InfoNCE on (query, tool) pairs from $\mathcal{D}_{\text{train}}$:

$$\theta_1 = \arg\min_\theta \mathbb{E}_{(q,t)\sim\mathcal{D}_{\text{train}}} \mathcal{L}_{\text{NCE}}\big(\theta; (q, \phi_5(t))\big) \tag{3}$$

This is the standard contrastive tool-retrieval recipe (Qin et al., 2024; Anantha et al., 2023; Shi et al., 2025), and is observed to be the strongest encoder-only baseline (Table 1). We initialise the loop from θ_1 rather than pretrained BGE so the encoder has a contrastive head start before description-only retraining begins.

S1b: Rewriter warmup. The rewriter is fine-tuned on the catalog itself, with each tool t shown under all five renderings $\phi_1, \ldots, \phi_5$ from the format family Φ (defined in §3.2):

$$\psi_1 = \arg\min_\psi -\sum_{t\in\mathcal{T}} \sum_{\phi_i\in\Phi} \log p_\psi\big(\phi_i(t)\big) \tag{4}$$

This teaches the rewriter the catalog's vocabulary, naming conventions, and the multiple surface forms a tool can take.

Algorithm 1 CoHyDE: Iterative Co-Training

Require: Pretrained encoder θ_0, base rewriter ψ_0, training pairs $\mathcal{D}_{\text{train}}$, prompt ρ_{HyDE}, rendering family Φ, rounds R

Ensure: Co-trained encoder θ_{R+1} and rewriter ψ_{R+1}

[S1a] *Encoder warmup:* train θ_0 with InfoNCE on $\{(q,\ \phi_5(t))\}$ from $\mathcal{D}_{\text{train}}$ to obtain θ_1
[S1b] *Rewriter warmup:* fine-tune ψ_0 on catalog tools under Φ to obtain ψ_1
[S2] *Bootstrap:* generate $\mathcal{D}_{\text{d}}^{(\psi_1)} = \{(g_{\psi_1}(\rho_{\text{HyDE}}(q)),\ \phi(t))\}$ for $(q,t) \in \mathcal{D}_{\text{train}}$
for $r = 1, \ldots, R$ **do**
  [S3$_r$] *Encoder retrain:* train θ_r with InfoNCE on $\mathcal{D}_{\text{d}}^{(\psi_r)}$ to obtain θ_{r+1}
  [S4$_r$] *Rewriter alignment:*
    Sample N descriptions $\{\tilde{d}^{(j)}\} \sim g_{\psi_r}(\rho_{\text{HyDE}}(q))$ for each $q \in \mathcal{D}_{\text{train}}$
    Score each $\tilde{d}^{(j)}$ by NDCG@5 under θ_{r+1}
    Form preference pair: $\tilde{d}_q^+ = \arg\max_j \text{NDCG@5}(\tilde{d}^{(j)})$, $\tilde{d}_q^- = \arg\min_j$
    $\psi_{r+1} \leftarrow \arg\min_\psi \mathcal{L}_{\text{DPO}}(\psi;\ \psi_r)$
    $\mathcal{D}_{\text{d}}^{(\psi_{r+1})} = \{(g_{\psi_{r+1}}(\rho_{\text{HyDE}}(q)),\ \phi(t))\}$
end for
return $(\theta_{R+1},\ \psi_{R+1})$

S2: Bootstrap data generation. Using ψ_1 and the prompt ρ_{HyDE}, we generate the first round of (description, tool) training data:

$$\mathcal{D}_{\text{d}}^{(\psi_1)} = \big\{(g_{\psi_1}(\rho_{\text{HyDE}}(q)), \phi(t)) : (q,t) \in \mathcal{D}_{\text{train}}\big\} \tag{5}$$

with $\phi \sim \text{Unif}(\Phi)$. The 5-format-trained rewriter produces catalog-style tool descriptions, used as the contrastive anchors for the next encoder training.

S3$_r$: Encoder retraining. For each round $r = 1, \ldots, R$, the encoder is trained further on $\mathcal{D}_{\text{d}}^{(\psi_r)}$, continuing from θ_r:

$$\theta_{r+1} = \arg\min_\theta \mathbb{E}\, \mathcal{L}_{\text{NCE}}\big(\theta; (g_{\psi_r}(\rho_{\text{HyDE}}(q)), \phi(t))\big) \tag{6}$$

No real (q,t) pair participates in this stage; the encoder is trained *only* on $(g_{\psi_r}(q), \phi(t))$ pairs.

S4$_r$: DPO alignment of the rewriter. For each $q \in \mathcal{D}_{\text{train}}$, sample N candidate descriptions $\{\tilde{d}^{(j)}\} \sim g_{\psi_r}(\rho_{\text{HyDE}}(q))$ at and score them by NDCG@5 under the just-trained encoder θ_{r+1}.

Form a preference pair $(\tilde{d}_q^+, \tilde{d}_q^-)$ from the argmax and argmin of those scores, and minimise the standard DPO objective (Rafailov et al., 2023):

$$\mathcal{L}_{\mathrm{DPO}}(\psi; \psi_r) = -\mathbb{E}_q \log \sigma \Bigg(\beta \log \frac{p_\psi(\tilde{d}_q^+ | \rho(q))}{p_{\psi_r}(\tilde{d}_q^+ | \rho(q))} - \beta \log \frac{p_\psi(\tilde{d}_q^- | \rho(q))}{p_{\psi_r}(\tilde{d}_q^- | \rho(q))} \Bigg) \quad (7)$$

$\psi_{r+1} = \arg\min_\psi \mathcal{L}_{\mathrm{DPO}}(\psi; \psi_r)$ is then used to regenerate $\mathcal{D}_{\mathrm{d}}^{(\psi_{r+1})}$ for the next round. The encoder of round r supervises the rewriter update, and the rewriter of round $r+1$ produces the data for the next encoder update, both sides evolve along a coupled trajectory.

Iteration. The loop $\{\mathrm{S3}_r, \mathrm{S4}_r\}$ may be unrolled for any number of rounds R.

3.6 Evaluation Protocol

We report hit@k, recall@k, and NDCG@k for $k \in \{1, 5, 10, 20\}$, averaged over each query split $\mathcal{Q} \in \{\mathcal{Q}_{\mathrm{eval}}, \mathcal{Q}_{\mathrm{vague}}\}$ and stratified by tier (G1/G2/G3). Catalog embeddings $\{f_\theta(\phi_5(t))\}_{t \in \mathcal{T}}$ are precomputed once per encoder θ under ϕ_5 and reused across query splits; rewriter outputs are regenerated end-to-end for every reported configuration. Metric definitions appear in Appendix I; the full k-sweep results in Appendix J.

4 Experiments & Results

4.1 Experimental Setup

Benchmark and evaluation splits. All experiments use the ToolBench-derived catalog and query splits described in §3.2: a 10,000-tool subset $\mathcal{T}$ with 1,092 evaluation queries stratified across three tiers (G1/G2/G3). Each query is evaluated on both the standard split $\mathcal{Q}_{\mathrm{eval}}$ — the original ToolBench queries — and the vague split $\mathcal{Q}_{\mathrm{vague}}$, which contains intent-preserving paraphrases that replace surface tokens with conversational alternatives (both splits share the same gold tool sets).

Baselines. We compare against seven reference points spanning the space of design choices. **BM25** over the ϕ_5-indexed catalog serves as a sparse lexical floor, requiring no training or LLM. **BGE (vanilla)** and **text-embedding-3-large** are frozen dense encoders that embed raw queries directly. **Query expansion (LLM + BGE)** and **HyDE (vanilla LLM + BGE)** both pair the same vanilla BGE encoder with the same vanilla Qwen3.5-4B generator, but differ in generation strategy: query expansion paraphrases the user query (anchor stays on the query side), while HyDE generates a hypothetical catalog-style tool description (anchor moves to the document side). **BGE (trained S1a)** is the BGE encoder fine-tuned on (query, tool) pairs at the S1a warmup step described in §3.5. **HyDE (vanilla LLM + trained BGE S1a)** pairs the trained encoder with HyDE generation without any rewriter training, testing whether the two components can be composed after independent optimisation. All baselines use the ϕ_5 catalog index for a fair comparison; all LLM-based baselines use Qwen3.5-4B (Yang et al., 2025) as the generator.

CoHyDE inference. At test time, the trained rewriter produces a hypothetical tool description $\tilde{d} = \mathrm{clean}(g_\psi(\rho_{\mathrm{HyDE}}(q)))$ via greedy decoding (temperature=0, 150-token budget). The trained encoder takes $\tilde{d}$ as its query and retrieves the top-k tools by nearest-neighbour lookup against the catalog indexed under ϕ_5. Full training hyperparameters and infrastructure details are in Appendix E.

Metrics. NDCG@5 is the primary metric; Recall@5 is reported as a secondary check that gains reflect more correct tools being retrieved and not merely reranking an already-correct candidate set. Both metrics are reported on $\mathcal{Q}_{\mathrm{eval}}$ and $\mathcal{Q}_{\mathrm{vague}}$, stratified by tier (G1 / G2 / G3), giving six (metric $\times$ split $\times$ tier) cells per configuration.

4.2 CoHyDE Comparison with Baselines

Table 1 compares CoHyDE against seven reference points; reading the rows top to bottom traces the logical sequence that motivates the co-training design.

Encoder-only fine-tuning is brittle on vague queries. The InfoNCE-trained encoder (BGE S1a) dominates every standard evaluation split by a wide margin, lifting G1 NDCG@5 from 56.5 to 84.2 over vanilla BGE. On vague paraphrases of the same queries, however, it collapses: G1 vague falls -39.5pp from its own performance on standard counterpart, and G3 vague reaches 14.9% — barely above the vanilla baseline. The strong commercial encoder (text-embedding-3-large) follows the same pattern at a lower absolute level: competitive on standard, but no more robust on vague. The encoder has learned a similarity function calibrated to the surface vocabulary of well-formed queries;

	Standard						Vague					
	G1		G2		G3		G1		G2		G3	
Method	N@5	R@5	N@5	R@5	N@5	R@5	N@5	R@5	N@5	R@5	N@5	R@5
BM25	49.0	52.7	31.0	31.7	17.3	17.5	17.8	19.6	10.3	11.6	3.4	3.0
BGE (vanilla)	56.5	62.4	31.9	33.2	18.5	20.7	33.2	37.3	15.8	16.7	8.3	12.9
text-embedding-3-large	61.6	66.4	36.3	36.4	26.7	29.2	34.8	40.5	18.1	19.8	19.8	19.8
Query expansion† (LLM + BGE)	51.4	56.7	27.0	30.0	29.2	30.9	32.6	38.2	11.3	12.5	6.2	10.9
HyDE (vanilla LLM & BGE)	58.2	63.8	34.9	37.7	35.1	36.4	37.4	43.4	17.7	19.9	17.4	21.1
BGE (trained S1a)	84.2	89.7	71.7	76.3	57.1	56.8	44.7	49.8	30.7	33.6	14.9	18.7
HyDE (vanilla LLM + trained BGE S1a)	73.4	79.5	59.5	62.2	46.2	47.6	41.2	47.4	29.3	31.8	17.6	17.9
CoHyDE ($r = 3$)	**86.8**	**91.0**	**73.6**	**78.0**	**60.1**	**60.4**	**49.4**	**55.2**	**38.7**	**41.5**	**21.1**	**26.2**

† Vanilla LLM paraphrases the user query into a retrieval-friendly form (query-side expansion); the rewritten query is encoded by vanilla BGE.

Table 1: NDCG@5 (N@5) and Recall@5 (R@5) in % on standard and vague query splits, stratified by tier. Bold = best per column.

any deviation from that vocabulary exposes its brittleness.

Description generation bridges vocabulary gaps; query rewriting does not. Table 1 includes both a query expansion baseline and a HyDE baseline, both using the same vanilla BGE encoder and the same vanilla Qwen3.5-4B generator. On standard queries the two are comparable; the decisive difference is on vague cross-domain queries. Query rewriting, which keeps the inference-time anchor on the query side of the embedding space, reaches G3 vague NDCG@5 of only 6.2%—below the vanilla BGE baseline of 8.3%. HyDE, which generates hypothetical catalog-style tool descriptions and moves the anchor to the document side, reaches 17.4% on the same split, a +11.2pp gap. The pattern is consistent across all tiers: HyDE outperforms query rewriting on every vague cell, often by double-digit margins. This establishes the generative direction that CoHyDE adopts: producing a hypothetical tool description rather than reformulating the query.

Combining HyDE with a query-trained encoder makes things worse. A natural next step is to combine the gains of encoder fine-tuning with HyDE generation. Table 1 shows that this naive combination *backfires*: "HyDE (vanilla LLM + trained BGE S1a)" drops −10.8pp on G1 standard NDCG@5 relative to the trained encoder used alone (73.4 vs 84.2), and trails on every other split as well. The trained encoder's similarity function was calibrated on raw user queries as anchors; at inference it receives hypothetical catalog descriptions whose embedding distribution is shifted away from that calibration manifold, distorting the nearest-neighbour search. This is the direct motivation for co-training: the encoder and rewriter cannot be composed after independent training. Instead, they should evolve their representation spaces together.

CoHyDE resolves all three failure modes simultaneously. CoHyDE at r=3 improves over the strongest single-component baseline (BGE S1a) on every split. Standard-query gains are modest (average +2.5pp), reflecting that co-training preserves the encoder's standard-query precision rather than trading it away. Vague-query gains are substantially larger (average +6.3pp), closing the lexical brittleness that neither the trained encoder nor baseline HyDE could resolve on its own. Crucially, the co-trained encoder also closes the representation-mismatch gap: trained exclusively on DPO-generated hypothetical descriptions with zero raw queries in its training data, it reaches G1 standard NDCG@5 of 86.8%, matching and slightly exceeding the BGE encoder trained on raw queries. The jointly-trained space has been shaped so that raw query vectors at inference land in the same neighbourhood as their corresponding catalog descriptions, without ever having seen those queries during training.

4.3 Ablations

We isolate four design choices in CoHyDE: (i) the rewriter warmup stage S1b, which pre-trains the LLM on catalog surface forms before the co-training loop begins; (ii) the joint encoder update, asking whether the gains require a co-trained encoder or can be obtained by pairing the trained rewriter with a vanilla encoder; (iii) the symmetric question for the encoder side, asking whether the co-trained encoder retains its advantage when

	Standard						Vague					
	G1		G2		G3		G1		G2		G3	
Variant	N@5	R@5	N@5	R@5	N@5	R@5	N@5	R@5	N@5	R@5	N@5	R@5
CoHyDE (full)	**86.8**	**91.0**	73.6	**78.0**	**60.1**	**60.4**	**49.4**	**55.2**	**38.7**	**41.5**	**21.1**	**26.2**
CoHyDE (w/o S1b rewriter warmup)	81.3	87.0	71.5	75.6	50.5	53.8	47.0	54.1	35.2	36.9	19.6	21.8
CoHyDE (trained LLM + vanilla encoder)	63.2	68.7	38.1	40.3	36.2	37.0	40.1	45.2	17.6	19.4	12.9	15.3
CoHyDE (vanilla LLM + trained encoder)	86.3	79.5	**75.6**	62.2	53.7	47.6	44.1	47.4	32.8	31.8	15.8	17.9

Table 2: Ablation study. Each row removes or replaces one component of CoHyDE. Bold = best per column.

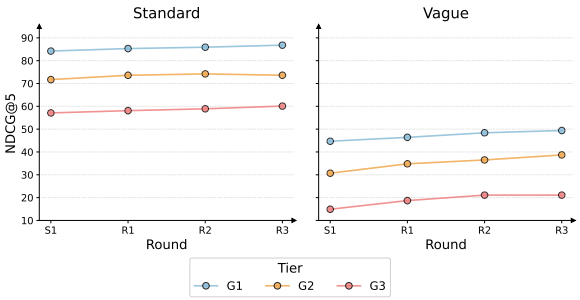

Figure 2: Per-round NDCG@5 trajectory on standard (left) and vague (right) query splits, stratified by tier. Both splits improve monotonically from S1 through R3.

paired with a vanilla (untrained) rewriter; and (iv) the number of co-training rounds r, which measures convergence behaviour and whether additional rounds continue to improve retrieval quality.

Table 2 reports results for each ablated variant across all six evaluation splits.

Rewriter warmup is critical for cross-domain retrieval. Removing the rewriter warmup drops standard G3 NDCG@5 by 9.6pp (60.1 $\rightarrow$ 50.5) and R@5 by 6.6pp, while standard G1 and G2 fall by only 5.5pp and 2.1pp respectively. Vague-query degradation is consistently smaller ($\leq$ 3.5pp across all tiers). The gradient of the drop, steepest on standard G3 and shallowest on vague splits, reflects what the warmup actually provides: the rewriter learns the catalog's vocabulary and surface forms *before* the co-training loop begins. On near-domain standard G1 queries, the encoder can partially compensate for a cold rewriter; on cross-domain standard G3 tools, whose descriptions share few surface tokens with user queries, a warmup-free rewriter fails to generate catalog-aligned descriptions from the outset and the encoder's nearest-neighbour search degrades from round one.

The trained rewriter requires a jointly-trained encoder. Pairing the co-trained rewriter with a vanilla BGE encoder produces the largest degradation in Table 2. NDCG@5 collapses on standard splits by 23.6pp, 35.5pp, and 23.9pp on G1, G2, and G3 respectively; vague splits decline by 9–21pp. The vanilla encoder was trained on raw user queries, so its representation space is calibrated to natural-language vectors rather than to the catalog-style hypothetical descriptions the DPO-aligned rewriter generates. Feeding it rewriter outputs at inference therefore distorts, rather than improves, the similarity search. This result confirms that the rewriter's gains are not a free add-on to any encoder: they require an encoder whose representation space has been co-shaped to match the rewriter's output distribution.

The encoder is load-bearing for standard queries; the rewriter differentiates vague ones. The symmetric ablation, co-trained encoder with a vanilla rewriter, reveals the complementary side. On easy standard queries, the co-trained encoder is nearly self-sufficient: G1 standard NDCG@5 falls by only 0.5pp (86.3 vs 86.8), and G2 standard actually edges out the full model by 2.0pp (75.6 vs 73.6). The co-trained encoder has absorbed enough of the catalog distribution that zero-shot HyDE queries land acceptably close in its embedding space without a fine-tuned rewriter. The gap opens on harder settings: NDCG@5 on standard G3 falls by 6.4pp (60.1 $\rightarrow$ 53.7) and on vague splits by 5.3–5.9pp uniformly across all tiers. These are precisely the conditions where the rewriter's DPO alignment matters—bridging a large lexical gap on cross-domain tools, or reasoning past underspecification on vague queries.

Together, ablations (ii) and (iii) confirm the asymmetry established in §4.2: the encoder carries precision on near-vocabulary standard queries; the rewriter provides robustness on hard and vague ones; co-training is what enables both gains simultaneously.

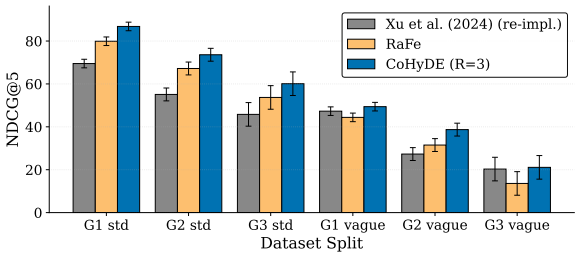

Figure 3: NDCG@5 comparison with the two closest prior methods across all six evaluation splits. Error bars show 95% confidence intervals.

Co-training performance evolution across rounds. Figure 2 traces NDCG@5 at each stage for all six evaluation splits. Performance is monotonically non-decreasing from S1 through R3 on five of six splits; the single exception is standard G2, which retreats by a marginal 0.6pp between R2 and R3. Gains from R1 to R2 are consistently larger than those from R2 to R3 across all tiers and both evaluation split types, indicating the coupled encoder–rewriter system approaches convergence within three rounds. The diminishing updates and the single non-monotonic cell motivate our choice to report R3 as the final CoHyDE configuration.

4.4 Comparison with Closest Prior Methods

CoHyDE is most directly related to two lines of work that also use iterative feedback to improve tool or document retrieval. Shao et al. (2023) propose an iterative loop in which the LLM's *downstream tool-usage success* is fed back to retrain the retriever; the retriever evolves across rounds but the query representation at inference is the raw user query and no rewriter component is trained. RaFe (Mao et al., 2024) trains a query rewriter with RL feedback from an external reranker in a general RAG setting; critically, the rewriter *paraphrases the user query* into a more retrieval-friendly form — it does not generate catalog-style hypothetical descriptions — so the inference-time anchor remains on the query side of the embedding space, and the encoder remains frozen throughout. Both methods train only one side of the retrieval pipeline and use a signal external to the encoder-rewriter pair rather than closing the loop directly through the retrieval objective.

Figure 3 reports NDCG@5 for all three methods across standard and vague splits. On standard queries CoHyDE leads on all three tiers by a wide margin over Shao et al. (2023): +17.3pp on G1, +18.5pp on G2, and +14.3pp on G3. RaFe (Mao et al., 2024) is a stronger standard-query competitor than Shao et al. (2023), closing much of the gap, but still trails CoHyDE by 6.9pp on G1, 6.4pp on G2, and 6.4pp on G3. The vague splits separate the methods more sharply. Shao et al. (2023) holds up on G1 and G2 vague (within $\leq$ 2.1pp of CoHyDE), but falls behind on G3. RaFe (Mao et al., 2024) degrades most severely on G3 vague, dropping to 13.6 NDCG@5 against CoHyDE's 21.1 — a 7.5pp gap on the hardest cross-domain vague split, compared to RaFe's 6.4pp deficit on the corresponding standard split. Confidence intervals for all reported differences follow the paired-bootstrap protocol (Appendix K); re-implementation details for Shao et al. (2023) are in Appendix M.

The pattern is consistent with the failure-mode framing of §4.2: an encoder trained on raw queries with a frozen rewriter (i.e. Shao et al. (2023)) and a rewriter trained against an external reranker with a frozen encoder (i.e. Mao et al. (2024)) both fail to bridge the lexical gap that vague cross-domain queries expose. CoHyDE's joint co-training, where the encoder's retrieval metric directly supervises the rewriter and the rewriter's outputs shape the encoder's representation space, sustains the advantage across both query distributions.

5 Conclusion

Contrastive encoder fine-tuning and HyDE-style description generation fail in complementary directions, and their naive composition makes things worse because their representation spaces have been calibrated to different input distributions. We introduce CoHyDE, an iterative co-training loop that resolves this by evolving the encoder and rewriter together: the encoder's NDCG@5 scores supervise the rewriter via DPO, and the rewriter's catalog-aligned outputs become the encoder's training anchors each round. Three rounds improve over the strongest single-component baseline on every evaluation cell, with average gains of +2.5pp NDCG@5 on standard queries and +6.3pp on vague ones. The asymmetric improvement is the direct consequence of the mechanism: the jointly-trained encoder learns a space where raw query vectors land near their corresponding catalog descriptions at inference — without ever seeing those queries during training — suggesting that for retrieval over idiosyncratic catalogs with underspecified queries, the encoder and rewriter are better treated as a single co-evolving system.

Limitations

All reported numbers are from a single training seed; the bootstrap confidence intervals in Appendix P characterise evaluation-set variance but not training-side variance, and multi-seed retrains were not run due to the per-round compute cost of each co-training loop (Appendix N). Experiments are conducted on a 10K-tool English subset of ToolBench, which is skewed toward consumer-facing RapidAPI REST endpoints; it remains to be seen whether the co-training gains transfer to enterprise catalogs, non-English queries, or function-call schemas that lack free-text descriptions. The vague-query split $\mathcal{Q}_{\text{vague}}$ is generated and validated by the LLMs used throughout the pipeline; though spot-checked by a human on 50 paraphrases (Appendix A), systematic biases shared between the generator and judge may go undetected. Finally, we benchmark against single-vector dense retrievers and BM25 but not against cross-encoder rerankers or sparse–dense hybrids; a comparison with such methods would require matched latency or FLOPs budgets, which we leave to future work.

Ethical Considerations

We conducted experiments within the provisions of the ACL Ethics Policy and relevant research-integrity guidelines. There are, to the best of our knowledge, no remaining ethical risks that have not been addressed.

References

Raviteja Anantha, Bortik Bandyopadhyay, Anirudh Kashi, Sayantan Mahinder, Andrew W Hill, and Srinivas Chappidi. 2023. Protip: Progressive tool retrieval improves planning. *Preprint*, arXiv:2312.10332.

Anonymous. 2026. ToolSense: A diagnostic framework for auditing parametric tool knowledge in LLMs. Under review.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. 2022. Inpars: Data augmentation for information retrieval using large language models. *Preprint*, arXiv:2202.05144.

Kunfeng Chen, Luyao Zhuang, Fei Liao, Juhua Liu, Jian Wang, and Bo Du. 2026. Tool retrieval bridge: Aligning vague instructions with retriever preferences via bridge model. *Preprint*, arXiv:2604.07816.

Xilun Chen, Kushal Lakhotia, Barlas Oguz, Anchit Gupta, Patrick Lewis, Stan Peshterliev, Yashar Mehdad, Sonal Gupta, and Wen-tau Yih. 2022. Salient phrase aware dense retrieval: Can a dense retriever imitate a sparse one? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 250–262, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4705–4726, Miami, Florida, USA. Association for Computational Linguistics.

Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith Hall, and Ming-Wei Chang. 2023. Promptagator: Few-shot dense retrieval from 8 examples. In *The Eleventh International Conference on Learning Representations*.

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, Toronto, Canada. Association for Computational Linguistics.

Sheryl Hsu, Omar Khattab, Chelsea Finn, and Archit Sharma. 2025. Grounding by trying: LLMs with reinforcement learning-enhanced retrieval. In *The Thirteenth International Conference on Learning Representations*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: few-shot learning with retrieval augmented language models. *J. Mach. Learn. Res.*, 24(1).

Jeff Johnson, Matthijs Douze, and Herve Jegou. 2021. Billion-Scale Similarity Search with GPUs . *IEEE Transactions on Big Data*, 7(03):535–547.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Yibin Lei, Yu Cao, Tianyi Zhou, Tao Shen, and Andrew Yates. 2024. Corpus-steered query expansion with large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2:*

Short Papers), pages 393–401, St. Julian's, Malta. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. 2023. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6385–6400, Singapore. Association for Computational Linguistics.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Richard James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. 2024. RA-DIT: Retrieval-augmented dual instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Elias Lumer, Vamse Subbiah, James Burke, Pradeep Basavaraju, and Austin Huber. 2025. 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 - Volume 3: ICAART*, pages 1180–1191. INSTICC, SciTePress.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore. Association for Computational Linguistics.

Shengyu Mao, Yong Jiang, Boli Chen, Xiao Li, Peng Wang, Xinyu Wang, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2024. RaFe: Ranking feedback improves query rewriting for RAG. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 884–901, Miami, Florida, USA. Association for Computational Linguistics.

Rui Meng, Ye Liu, Semih Yavuz, Divyansh Agarwal, Lifu Tu, Ning Yu, Jianguo Zhang, Meghana Bhat, and Yingbo Zhou. 2024. Augtriever: Unsupervised dense retrieval and domain adaptation by scalable data augmentation. *Preprint*, arXiv:2212.08841.

Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. *Preprint*, arXiv:1904.08375.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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. 2024. 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 *CIKM*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Christopher Sciavolino, Zexuan Zhong, Jinhyuk Lee, and Danqi Chen. 2021. Simple entity-centric questions challenge dense retrievers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6138–6148, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8371–8384, Mexico City, Mexico. Association for Computational Linguistics.

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, Vienna, Austria. Association for Computational Linguistics.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2019. Representation learning with contrastive predictive coding. *Preprint*, arXiv:1807.03748.

Kexin Wang, Nandan Thakur, Nils Reimers, and Iryna Gurevych. 2022. GPL: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2345–2360, Seattle, United States. Association for Computational Linguistics.

Liang Wang, Nan Yang, and Furu Wei. 2023. Query2doc: Query expansion with large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

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*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649, New York, NY, USA. Association for Computing Machinery.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Yue Yu, Chenyan Xiong, Si Sun, Chao Zhang, and Arnold Overwijk. 2022. COCO-DR: Combating the distribution shift in zero-shot dense retrieval with contrastive and distributionally robust learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1462–1479, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Appendix

A Vague-Query Construction and Validation

The vague-query split $\mathcal{Q}_{\text{vague}}$ is a held-out paraphrase of the official 1,092-query evaluation set, constructed to probe robustness under query-side distribution shift while preserving the gold tool set. Construction follows the protocol of Chen et al. (2026) verbatim, with claude-4.5-opus substituted for their GPT-4o paraphraser.

Two-pass validation. The split is validated in two passes.

1. **LLM self-check.** Every paraphrase in $\mathcal{Q}_{\text{vague}}$ is re-presented to claude-4.5-opus in a separate session, together with the original query and the gold tool set, and scored on the three binary criteria of Chen et al. (2026) — (i) intent preservation, (ii) absence of leaked tool names / API verbs / domain keywords, (iii) plausibility as an end-user utterance. An example is retained only if all three criteria are satisfied. Substituting claude-4.5-opus for the GPT-4o validator used by Chen et al. (2026) is the only deviation from their protocol.

2. **Human spot-check.** 50 paraphrases were sampled uniformly at random from the LLM-validated split and re-verified by human against the same three criteria. All 50 passed all three criteria, giving a 6% rule-of-three upper bound on the true failure rate at 95% confidence. The annotator was not blinded to the paraphraser identity; this is a transparency disclosure rather than a methodological strength. The annotator was not compensated separately, apart from their regular wages during the research; no external annotators were used.

Ethics review. The annotation involves no human subjects beyond the human conducting the spot-check and qualifies as exempt from formal ethics-board review under the relevant institutional guidelines. No consent procedure was required because the annotator is the data producer.

Annotator demographics. The annotator is a full-time employee of the authoring organization and resides in the USA.

B Cleaning Operator

The deterministic cleaning operator $\text{clean}(\cdot)$ applied to every rewriter output before encoding strips:

1. Reasoning-trace blocks delimited by `<think>...</think>`.

2. Unclosed reasoning traces (a leading `<think>` with no terminator), in which case the entire output is rejected and replaced with the original query.

3. Conversational preambles matching `^(Sure|Okay|Of course|Here is|Here's)[^.]*\.\s+`.

4. Trailing whitespace and repeated blank lines.

The operator is implemented as a sequence of regular-expression substitutions and is applied identically at SFT-target construction, DPO-candidate scoring, and inference time.

C HyDE-Style Rewriter Prompt

The HyDE-style prompt ρ_{HyDE} is used at the optional SFT stage (when included; see Appendix H), at S2 (generating $\mathcal{D}_{\mathrm{d}}^{(\psi_r)}$), at S4 (sampling DPO candidates), and at every inference-time HyDE evaluation reported in §4.

System message:

> You are an expert at understanding API tool pipelines. When given a user query, you describe the sequence of API calls needed to fulfill it. Each description should focus on what the tool does, what inputs it takes, and what data it returns. Write each tool's description as a single concise technical sentence.

User message:

```
User query: {query}
Think about the full pipeline of API calls needed to answer the query. Describe each API tool in the pipeline in order, explaining what data it provides and how it feeds into the next step. Be concise and technical.
```

D Query-Rewriting Prompt

The query-rewriting prompt ρ_{rewrite} is used *only* in the prompt-style ablation reported in Appendix L; it is not used anywhere for CoHyDE.

System message:

> You are a query enhancement expert. Given a user query and the relevant API tools, rewrite the query to be more specific and detailed. Include relevant tool names, API capabilities, and technical terms that would help a retrieval system find the right tools. Keep it as a natural user request, but a more specific version of what the user is asking for.

User message:

```
Original query: {query}
Relevant tools: {tool_names}
Rewritten query:
```

E Per-Stage Hyperparameter Summary

Table 3 consolidates every training and inference stage in the main pipeline with its load-bearing hyperparameters. Full per-stage detail (objective, optimiser, schedules, ablation context) is in Appendix F (encoder; S1a, S3$_r$) and Appendix G (rewriter; S1b, S2, S4$_r$). The optional HyDE-style SFT bridging stage — which is *not* part of the main pipeline — is documented separately in Appendix H. Software versions for every stage are in Appendix O.

F Encoder Training Hyperparameters

The encoder is trained at two distinct points in the pipeline: once at S1a (warmup on real query–tool pairs) and once per round at S3$_r$ (retrain on rewriter-generated descriptions). Both stages use the same InfoNCE objective; they differ only in the anchor source and in whether they continue from the previous checkpoint or restart from θ_0.

InfoNCE loss. Let $\mathcal{B} = \{(a_i, p_i)\}_{i=1}^{B}$ be a mini-batch of (anchor, positive) pairs, and write $S_{ij}^{\theta} = \langle f_\theta(a_i), f_\theta(p_j)\rangle/\tau$. The symmetric InfoNCE loss (van den Oord et al., 2019) is

$$\mathcal{L}_{\mathrm{NCE}}(\theta;\mathcal{B}) = -\frac{1}{2B}\sum_{i=1}^{B}\Bigg[\log\frac{\exp S_{ii}^{\theta}}{\sum_{j=1}^{B}\exp S_{ij}^{\theta}} + \log\frac{\exp S_{ii}^{\theta}}{\sum_{j=1}^{B}\exp S_{ji}^{\theta}}\Bigg], \quad (8)$$

with temperature $\tau = 0.05$. Negatives are in-batch (no hard-negative mining).

S1a: Encoder warmup. Anchors are real queries q, positives are $\phi_5(t)$ for the gold tool. Initialised from BGE-large-en-v1.5. AdamW with learning rate $\eta_\theta = 2 \times 10^{-5}$, weight decay 10^{-2}, cosine schedule with 5% warmup, batch size $B = 256$, max sequence length 256 tokens, 5 epochs over the 104,224 (G1+G2+G3) training pairs of $\mathcal{D}_{\mathrm{train}}$. Validation NDCG@5 (mean over G1/G2/G3 dev splits) is computed every 200 steps and the checkpoint maximising it is retained — this is step 3600 in our run. All training is on a single H200 GPU with native bf16 mixed precision; no gradient accumulation. CLS-token pooling, L_2-normalised before scoring. No dropout beyond BGE's defaults.

S3$_r$: Per-round encoder retrain. Anchors are the rewriter outputs $\tilde{d} = g_{\psi_r}(\rho_{\mathrm{HyDE}}(q))$ from the

Stage	Objective	Init from	LR / β	Effective BS	Length / steps	Adapter
S1a (encoder)	InfoNCE, τ=0.05	BGE-large-en-v1.5	2×10^{-5}	256	seq 256, 5 ep	full FT
S1b (rewriter)	5-format SFT (NLL)	Qwen3.5-4B	2×10^{-5}	64	seq 1024, 8 ep	LoRA r=16 + new tok embeds
S2 (gen)	greedy decode, T=0	ψ_1	—	1	150-tok budget	—
S3$_r$ (encoder)	InfoNCE, τ=0.05	θ_r	2×10^{-5}	256	seq 256, 5 ep	full FT
S4$_r$ (cand. sample)	nucleus, T=0.7, p=0.95	ψ_r	—	N=4/query	300-tok budget	—
S4$_r$ (DPO)	sigmoid DPO, β=0.1	ψ_r (ref)	5×10^{-6}	8	prompt 1024 / comp 300, 1 epoch	LoRA r'=64
Inference	greedy decode, T=0	ψ_R	—	1	150-tok budget	—

Table 3: Per-stage hyperparameters for the main pipeline. "LR" is the optimiser learning rate (AdamW, weight decay 10^{-2}, bf16 throughout); for S4 "β" is the DPO regularisation coefficient. "Effective BS" is per-device batch size $\times$ gradient accumulation. All training runs on a single H200 GPU; full per-stage detail in Appendices F, G.

regenerated bootstrap set $\mathcal{D}_{\mathrm{d}}^{(\psi_r)}$; positives are $\phi(t)$ for $\phi \sim \mathrm{Unif}(\Phi)$. The encoder is initialised from θ_r (i.e. continued from the previous round's checkpoint, not from θ_0). All other hyperparameters — optimiser, learning rate 2×10^{-5}, weight decay, cosine schedule with 5% warmup, batch size 256, max sequence length 256, bf16, validation cadence, single-GPU — are identical to S1a. The retrain runs for the same 5-epoch budget over $\mathcal{D}_{\mathrm{d}}^{(\psi_r)}$, with the best validation-NDCG@5 checkpoint retained (around step 3400–4000 across rounds). *No real (q, t) pair is used at S3$_r$;* the encoder is trained purely on (rewritten-description, tool) pairs and tested on real queries at inference time. An ablation that mixes real q-anchored and $\tilde{d}$-anchored pairs in the same retrain is reported in Appendix L (combined-pair encoder retrain).

G Rewriter Training and Inference Hyperparameters

The rewriter is trained at S4$_r$ (DPO alignment, run once per round), and is sampled from at S2 (bootstrap data generation), at S4$_r$ (DPO candidate sampling), and at inference time. Each of these uses different decoding settings, listed below.

S1b: 5-format tool-rendering SFT. The rewriter ψ_0 = Qwen3.5-4B is fine-tuned on the catalog $\mathcal{T}$ rendered under all five formats $\phi_1, \ldots, \phi_5$ (defined in §3.2). Each tool is presented as a next-token prediction target under each of the five rendering conventions, sampled with equal weight per mini-batch. LoRA with rank $r = 16$, $\alpha_{\mathrm{LoRA}} = 32$, dropout 0.05, applied to attention q, k, v, o projections. AdamW with learning rate $\eta_\psi^{\mathrm{S1b}} = 2 \times 10^{-5}$, linear schedule with 3% warmup, per-device batch size 2, gradient accumulation 32 (effective batch size 64), max sequence length 1024, 8 epochs over the mixture ($\approx$50K examples per epoch), bf16 mixed precision, gradient checkpointing on (non-reentrant), single H200 GPU. Validation hit@5 on the G1/G2/G3 retrieval dev splits is computed every 100 steps and the best checkpoint is retained.

S2: Bootstrap description generation. Using ψ_1 (the S1b checkpoint), we generate the first round of (description, tool) training data $\mathcal{D}_{\mathrm{d}}^{(\psi_1)}$ over all queries $q \in \mathcal{D}_{\mathrm{train}}$. Sampling is greedy ($T = 0$, top-$p = 1$, top-$k = 1$, no repetition penalty) with a 150-token completion budget, served via vLLM. We use a single completion per query. Outputs pass through $\mathrm{clean}(\cdot)$ (Appendix B) before being used as encoder anchors. The same generation protocol is re-run at the start of every subsequent round r to produce $\mathcal{D}_{\mathrm{d}}^{(\psi_r)}$ from the current rewriter ψ_r.

S4$_r$: DPO candidate sampling. For each query $q \in \mathcal{D}_{\mathrm{train}}$ we sample $N = 4$ candidate descriptions from ψ_r at temperature $T = 0.7$, top-$p = 0.95$, top-$k = 50$, with a 300-token completion budget, served via vLLM. Each candidate $\tilde{d}^{(j)}$ is encoded by the freshly-retrained encoder θ_{r+1} (from S3$_r$); candidates are scored by their NDCG@5 against the gold tool set T_q^* under θ_{r+1}. The chosen / rejected pair $(\tilde{d}_q^+, \tilde{d}_q^-)$ is the (argmax, argmin) of the four scores. Queries whose four candidates yield identical NDCG@5 are dropped from the DPO set.

S4$_r$: DPO training. We use TRL's `DPOTrainer` with the sigmoid loss formulation (Rafailov et al., 2023):

$$\mathcal{L}_{\mathrm{DPO}}(\psi; \psi_r) = -\log \sigma\Big(\beta\big[\Delta_\psi(\tilde{d}^+, q) - \Delta_\psi(\tilde{d}^-, q)\big]\Big),$$

with $\Delta_\psi(\tilde{d}, q) = \log \frac{p_\psi(\tilde{d}|\rho_{\mathrm{HyDE}}(q))}{p_{\psi_r}(\tilde{d}|\rho_{\mathrm{HyDE}}(q))}$ and $\beta = 0.1$. LoRA with rank $r' = 64$, $\alpha_{\mathrm{LoRA}} = 128$, dropout 0.05, applied to attention q, k, v, o projections only; embeddings and the language-model head are *not* tuned at S4 (the new tool tokens are already learned at S1b and held fixed thereafter). AdamW with learning rate $\eta_\psi^{\mathrm{S4}} = 5 \times 10^{-6}$, cosine schedule with

3% warmup, per-device batch size 2, gradient accumulation 4 (effective batch size 8), max prompt length 1024 tokens, max completion length 300 tokens, 1 epoch over the DPO pair set (≈4,371 optimiser steps), bf16 mixed precision, gradient checkpointing on, single H200 GPU. The reference policy ψ_r is the previous round's rewriter; at $r = 1$ this is ψ_1 from S1b. The trained adapter is merged back into the base weights at the end of the round before ψ_{r+1} is used at S2 of round $r + 1$.

Inference-time decoding. At evaluation time the rewriter is sampled greedily ($T = 0$) with a 150-token completion budget, single completion per query, served via vLLM. The deterministic cleaning operator $\mathrm{clean}(\cdot)$ (Appendix B) is applied before the description is passed to the encoder. The same decoding protocol is used for both standard and vague evaluation passes.

Reference policy and adapter merging. At each round r, the DPO reference ψ_r is loaded from the merged checkpoint of round $r-1$ (or from ψ_1 at $r = 1$). After DPO training, the LoRA adapter is merged into the base weights to produce ψ_{r+1}, which serves both as the next round's S2 generator and as the next round's DPO reference.

Implementation. PyTorch with HuggingFace Transformers; encoder training uses an in-house InfoNCE script; rewriter SFT and DPO use TRL's `SFTTrainer` and `DPOTrainer` with PEFT for LoRA adapters. vLLM serves the rewriter at S2, S4 candidate sampling, and inference. Exact software versions are listed in Appendix O.

H Optional SFT Stage (HyDE-Style Bridging)

This appendix describes an *optional* HyDE-style SFT pass that we ran in early experiments but *do not* use in the main pipeline reported in §4. It is a separate stage from the 5-format tool-rendering SFT (S1b) used in the CoHyDE pipeline.

In early experiments we inserted a brief LoRA-SFT pass between S1 and $S2_0$ to align the rewriter's output style with the catalog rendering. The motivation was that the base Qwen3.5-4B rewriter under ρ_{HyDE} produced free-form text whose length and style differed visibly from the 5-format catalog rendering — in particular, outputs were often substantially longer than any single ϕ_i. A short SFT pass on cleaned descriptions taught ψ the catalog-style output vocabulary and stop tokens, narrowing this style gap.

Once we adopted the 5-format encoder warmup (S1) and the 5-format rewriter warmup S1b (Appendix G), the picture changed. By training the encoder under $\phi \sim \mathrm{Unif}(\Phi)$ — with ϕ_5 in particular being the long, multi-sentence rendering closest in length and style to the rewriter's output — the encoder learns a representation that is approximately invariant across the style gap this SFT pass was originally designed to close, and S1b then teaches the rewriter the catalog vocabulary directly. In this configuration, S2 can be initialised from ψ_1 (the S1b checkpoint) without an intervening HyDE-style SFT pass, and the iterative loop proceeds as in Algorithm 1.

Pipeline with optional SFT. When included, the optional SFT pass produces an alternative ψ_1 as follows. Sampling descriptions from ψ_0 under ρ_{HyDE} for queries in $\mathcal{D}_{\mathrm{train}}$ and applying $\mathrm{clean}(\cdot)$ with a length filter ($|\tilde{d}| > 30$ characters) yields a cleaned set $\mathcal{S}_{\mathrm{SFT}}$ of ≈2,754 pairs. We then run a *short* LoRA SFT pass — explicitly *not* trained to convergence:

$$\psi_1^{(\mathrm{opt})} = \arg\min_{\psi} - \sum_{(q,\tilde{d})\in\mathcal{S}_{\mathrm{SFT}}} \log p_\psi\big(\tilde{d} \,\big|\, \rho_{\mathrm{HyDE}}(q)\big). \tag{9}$$

The iterative loop then runs from $\psi_1^{(\mathrm{opt})}$ instead of ψ_1 from S1b.

Hyperparameters. LoRA with rank $r = 16$, $\alpha_{\mathrm{LoRA}} = 32$, dropout 0.05, applied to all attention projection matrices (q, k, v, o); embeddings and the language-model head are not tuned. AdamW with $\eta_\psi = 2 \times 10^{-4}$, linear schedule with 20-step warmup, effective batch size 32 (per-device 4 × gradient accumulation 8), max sequence length 512, 100 optimisation steps total (≈3,200 examples seen, less than 2 epochs over $\mathcal{S}_{\mathrm{SFT}}$). bf16 mixed precision, single H200 GPU, gradient checkpointing on. Longer schedules (1,000 / 5,000 steps) degraded downstream DPO performance by reducing the diversity of candidates available to the S4 sampler; this ablation is reported in Appendix L.

Ablation. §4 reports retrieval numbers for the main pipeline (without the optional SFT pass, i.e. S2 initialised from S1b). The variant with the optional SFT pass is reported in Appendix L and does not improve over the main pipeline.

Metric	Standard G1	Standard G2	Standard G3	Vague G1	Vague G2	Vague G3
hit@1	83.1	75.2	69.0	44.7	39.4	18.0
hit@5	95.8	90.7	92.0	66.1	65.2	58.0
hit@10	97.3	92.0	96.0	75.7	73.9	74.0
hit@20	98.0	93.0	98.0	82.1	80.0	86.0
recall@1	40.4	32.9	25.2	21.8	16.9	6.8
recall@5	91.0	78.0	60.4	55.2	41.5	26.2
recall@10	95.2	86.6	75.4	67.2	54.3	42.2
recall@20	96.7	89.6	90.5	76.1	65.0	57.8
NDCG@1	83.1	75.2	69.0	44.7	39.4	18.0
NDCG@5	86.8	73.6	60.1	49.4	38.7	21.1
NDCG@10	87.8	78.3	66.8	54.4	44.0	28.0
NDCG@20	88.3	78.4	72.4	57.2	47.5	33.6

Table 4: Full k-sweep for the round-3 co-trained system. NDCG@1 = hit@1 by construction. Recall@1 is capped at $1/|T_q^*|$ for multi-tool queries.

I Evaluation Metrics

For a query q with gold tool set T_q^* and retrieved ranking $\hat{T}_k(q) = (\hat{t}_1, \ldots, \hat{t}_k)$:

$$\text{hit@}k(q) = \mathbb{1}\big[\hat{T}_k(q) \cap T_q^* \neq \varnothing\big], \tag{10}$$

$$\text{recall@}k(q) = \frac{|\hat{T}_k(q) \cap T_q^*|}{|T_q^*|}, \tag{11}$$

$$\text{NDCG@}k(q) = \frac{\sum_{j=1}^{k} \frac{\mathbb{1}[\hat{t}_j \in T_q^*]}{\log_2(j+1)}}{\sum_{j=1}^{\min(k,|T_q^*|)} \frac{1}{\log_2(j+1)}}. \tag{12}$$

Each metric is averaged over queries in the relevant tier. Definitions match the standard `ir_measures` implementations.

J Round-3 k-Sweep

Table 4 reports hit@k, recall@k, and NDCG@k for the converged round-3 co-trained system at $k \in \{1, 5, 10, 20\}$, on both standard and vague query splits, stratified by tier. Numbers are sourced from the same evaluation run that supplies the round-3 NDCG@5. NDCG@1 equals hit@1 by construction. Recall@1 is reported in full but, as noted in §3.6, is bounded above by $1/|T_q^*|$ and is therefore lower than the other metrics for every multi-tool query.

K Bootstrap CI Protocol

The paired-bootstrap 95% confidence intervals reported in §4 and Appendix P are computed as follows. For each tier $G \in \{G_1, G_2, G_3\}$ and split $\in \{\text{standard}, \text{vague}\}$, let $\{x_q\}_{q \in \mathcal{Q}^{(G)}}$ and $\{y_q\}_{q \in \mathcal{Q}^{(G)}}$ be the per-query NDCG@5 scores under the two systems being compared (e.g. Round 3 and Xu re-implementation), each system having produced its own ranking for the same set of queries from the same evaluation run. We resample query indices with replacement, $B = 10{,}000$ times, with a fixed random seed. For each resample b, we compute $\bar{x}^{(b)} = \text{mean}_{q \in S_b} x_q$ and $\bar{y}^{(b)} = \text{mean}_{q \in S_b} y_q$, and the paired difference $\delta^{(b)} = \bar{x}^{(b)} - \bar{y}^{(b)}$. The 95% CI of the difference is the (2.5%, 97.5%) percentile interval of $\{\delta^{(b)}\}_{b=1}^{B}$; we report this as $[\text{lo}, \text{hi}]$. The same protocol with a single-system x-only resample yields the per-method CI half-widths quoted in Appendix P ($\pm$2pp / $\pm$3pp / $\pm$5$-$6pp on G1/G2/G3, dominated by tier size $|\mathcal{Q}^{(G)}|$).

L Design-Choice Ablations: Details

This appendix gives the per-variant numbers and discussion behind the summary in §4.3. None of these variants are part of the main pipeline.

Single-format encoder training ($\phi \equiv \phi_i$ for a fixed i). Training under any single rendering matched the 5-format encoder on its matched evaluation rendering but underperformed on the others. Training under ϕ_5 alone — the rendering closest in length to the rewriter's output — still produced an encoder less robust to rewriter outputs of varying length than the 5-format encoder. We interpret this as evidence that the format mixture is doing more than augmenting on the longest format: by forcing the encoder to assign similar embeddings to the same tool across five different surface forms, it learns a length- and style-invariant representation that the description-only S2 retrains can then build on.

Combined-pair encoder retrain. Replacing the description-only S3$_r$ objective with the mixed batch $\mathcal{D}_{\text{q+d}}^{(\psi_r);\alpha=0.5}$ produced no improvement over description-only and slightly degraded vague-query performance. The mechanism we attribute this to is that mixing q-anchored pairs back into S3 partially pulls the encoder toward the on-distribution q-anchored fixed point established at S1 — the very fixed point whose vague-query failure we are trying to escape. The description-only objective is, in this view, doing distribution shift on purpose.

Query-rewrite prompt ρ_{rewrite}. Substituting the catalog-style ρ_{HyDE} with the user-style ρ_{rewrite} at

any stage of the loop (sampling SFT targets, generating S3$_r$ pairs, or sampling DPO candidates) lost on every standard metric, with the largest gap on cross-domain G3. The prompt is given the relevant tool names as in-context anchors, so the comparison is not a strawman: with that anchoring, ρ_{rewrite} produces specific, plausible user queries — they simply do not match the style of the contrastive pair the encoder sees during S3 retraining.

Longer SFT schedules. Extending the optional SFT pass (Appendix H) from 100 steps to 1,000 / 5,000 steps closed the SFT train loss but produced lower-diversity candidates at S4 and a smaller DPO margin, ultimately reducing the closed-loop gain. The DPO update relies on temperature-0.7 sampling spreading mass across distinguishable candidates; over-fitted rewriters concentrate that mass and degrade the preference signal.

HyDE-concat. Concatenating q with $\tilde{d}$ before encoding, in the spirit of Query2doc (Wang et al., 2023), helped slightly on G1 standard but hurt vague queries — where the original q's lexical surface is precisely the surface we are trying to escape.

M Xu et al. 2024 Re-implementation

This appendix documents the hyperparameters and prompts used for the head-to-head against Shao et al. (2023) reported in §4.4.

Encoder. Shao et al. (2023)'s pipeline trains the encoder once contrastively and never updates it again at inference time. We instantiate this once-trained encoder with our S1a InfoNCE checkpoint (BGE-large-en-v1.5 fine-tuned with InfoNCE on real $(q, \phi_5(t))$ pairs; full hyperparameters in Appendix F). This is a strictly stronger starting point than the Sentence-BERT base used in the original paper, and is therefore a charitable substitution: any gap our co-training closes against this baseline cannot be attributed to a weaker re-implemented encoder.

LLM refiner. Qwen3.5-4B served via vLLM, identical model and serving setup as the main paper's rewriter (we deliberately use the same LLM as our rewriter to remove model-capacity confounds; Shao et al. (2023) use GPT-3.5). Greedy decoding ($T = 0$, top-$p = 1$, top-$k = 1$), max 400 generated tokens per stage. The same temperature is used at all three prompted stages within a round.

Iteration schedule. $T = 3$ refinement rounds (matching the paper's reported best). Within each round, the LLM sees the current top-K retrieved tools with $K = 10$ and runs the three-stage Comprehension / Assessment / Refinement prompts; the refined instruction (or N/A early-stop) becomes the next round's retrieval input. Final ranking is the last round's. Top-50 are saved for evaluation at $k \in \{1, 5, 10, 20\}$.

Three-stage prompts. The paper does not provide verbatim text. Our reimplementation uses prompts that match the three-stage description in their §3:

- **P_Comprehension** (system + user): summarise user goals and the functionalities of the top-K retrieved tools, one short sentence per goal and per tool.
- **P_Assessment** (system + user): given the comprehension and the retrieved set, decide which goals are SOLVED vs UNSOLVED and whether the ranking matches importance; output `Solved:` and `Unsolved:` sections.
- **P_Refinement** (system + user): given the assessment, output either `N/A` (if all goals solved and ranking matches) or a refined one-paragraph instruction enriched with the missing intent.

This should be read as a faithful reimplementation of the pipeline structure rather than an exact reproduction of Xu's prompts.

Caveat. Beyond the prompt approximation, our reimplementation differs from the original paper in two respects: (i) a stronger encoder (S1a InfoNCE BGE-large vs Sentence-BERT base), and (ii) a different LLM (Qwen3.5-4B vs GPT-3.5). Both substitutions advantage Xu's method on this benchmark, making the head-to-head charitable to it.

N Compute Budget and Infrastructure

Hardware. All experiments were run on a single node with 8 H200 GPUs. The encoder training, rewriter SFT, rewriter DPO, and HyDE inference passes each fit on a single GPU; multi-GPU parallelism was used only opportunistically and not required for any reported result.

Per-stage wall-clock cost (approximate, single H200).

- S1a (encoder InfoNCE warmup, 5 epochs, batch 256): ∼3 hours.
- S1b (rewriter 5-format tool-memorisation SFT, ≈50K examples): ∼2 hours.
- Per-round S2 (description regeneration over $\mathcal{D}_{\text{train}}$ at $T = 0$, 150-token budget, via vLLM): ∼2 hours.
- Per-round S3 encoder retrain: ∼1.5 hours.
- Per-round S4 DPO data generation ($N = 4$ candidates per query at $T = 0.7$, scored by the current encoder): ∼6 hours.
- Per-round S4 DPO training (≈4,371 steps, LoRA $r = 64$): ∼4 hours.
- Per-configuration vague-split evaluation (HyDE generation over 1,092 queries via vLLM): ∼2.5 hours per pass.

Total budget. Three rounds of co-training plus all baselines, ablations, and rejected design-choice variants (Appendix L) totalled approximately 400–500 GPU-hours on H200-class hardware. Reproducing only the main result (S1a + S1b + three rounds + a single end-to-end vague evaluation) would take roughly 50 GPU-hours.

O Software Versions

Encoder training uses an in-house InfoNCE script built on PyTorch 2.4 and HuggingFace Transformers 4.46. Rewriter SFT and DPO use TRL 0.11 (`SFTTrainer`, `DPOTrainer`) with PEFT 0.13 for LoRA adapters. Rewriter inference uses vLLM 0.6. Mixed-precision training uses native PyTorch bf16. Evaluation uses our own retrieval scoring code; metric definitions match the standard `ir_measures` implementations and are given in closed form in Appendix I.

P Single-Seed Caveat

All reported numbers are from a single training seed. We did not run multi-seed variance estimates due to the per-round compute cost (§N); the per-round trajectory in §4.3 serves as a partial proxy for stability, in that the system's behaviour across rounds is smooth on tier-averaged metrics and only mildly non-monotonic at the per-cell level. As a separate, finite-sample uncertainty estimate, we computed paired-bootstrap 95% CIs (B=10,000) of NDCG@5 over the 593 / 399 / 100 queries in G1/G2/G3 (protocol in Appendix K). The half-width of these CIs — which captures sampling uncertainty over the eval set, *not* training-seed variance — is approximately ±2pp on G1 cells, ±3pp on G2 cells, and ±5–6pp on G3 cells (the smallest tier). Cell-level differences should accordingly be read against the bootstrap CI of the difference rather than a flat noise band: round-3 vs. S1 differences on standard tiers are well outside this band on G1/G2 and at the edge on G3; vague-tier differences are well outside on G2 but smaller than the bootstrap CI on G1 and G3. Multi-seed retrains, which would also bound training-side variance, are an open item.

Q Ethics, Risks, and Artifacts

Q.1 Upstream Artifacts and Licenses

This work builds on the following publicly available artifacts, used in a manner consistent with their stated intended use (research benchmarks and research-grade pretrained models).

- **ToolBench** (Qin et al., 2024): source of the underlying API pool and the official G1/G2/G3 evaluation queries. Released under Apache 2.0. https://github.com/OpenBMB/ToolBench.
- **ToolGen** (Wang et al., 2025): source of the 46,980-tool catalog from which we derive the 10K subset $\mathcal{T}$, and the source of the (query, gold-tool-set) training pairs $\mathcal{D}_{\text{train}}$. Released under Apache 2.0. https://github.com/Reason-Wang/ToolGen.
- **BGE-large-en-v1.5** (Xiao et al., 2024): encoder initialisation θ_0. Released under MIT. https://huggingface.co/BAAI/bge-large-en-v1.5.
- **Qwen3.5-4B** (Yang et al., 2025): rewriter initialisation ψ_0. Released under Apache 2.0. https://huggingface.co/Qwen/Qwen3.5-4B.

Q.2 Data Coverage and Privacy

Language and domain. The ToolBench/ToolGen catalog is entirely English-language and is sourced from RapidAPI's public catalog, skewed toward consumer-facing REST

APIs (weather, sports, lifestyle, finance, entertainment). No non-English text appears in queries, tool descriptions, or rewriter outputs.

Personally identifying information. Tool records contain API metadata (titles, endpoints, parameter schemas, free-text descriptions written by API publishers). They do not contain end-user PII. We did not run a dedicated PII scan on the catalog because the source records are already public API documentation; however, the manual review of 100 vague-paraphrase outputs in Appendix A did not surface any inadvertent generation of personal information.

Offensive content. The catalog includes some adult-content-tagged APIs (a small minority, consistent with RapidAPI's public listings). We did not filter these out, on the grounds that doing so would change the benchmark composition and make our numbers incomparable with prior work on the same catalog. No offensive content appears in any reported example or figure.

Split sizes. As reported in §3: catalog $|\mathcal{T}| = 10{,}000$; training set $|\mathcal{D}_{\text{train}}| = 104{,}224$ (G1: 44,873; G2: 35,402; G3: 23,949); evaluation queries 593 / 399 / 100 over G1/G2/G3 (1,092 total), with vague paraphrases of the same evaluation queries forming $\mathcal{Q}_{\text{vague}}$ of equal size.

Q.3 Risks

Tool retrieval is a component of larger tool-using agent systems; improvements in retrieval can amplify both desirable and undesirable downstream agent behaviour, depending on the tools in the catalog and the agent's policy over them. Our experiments are run on the ToolBench-derived ToolGen catalog, which inherits whatever selection biases that catalog has — consumer-facing REST APIs over enterprise or safety-critical tools, English-language descriptions, no audit of the underlying APIs' content. A rewriter aligned to a specific encoder is, in effect, a steering vector over that encoder's retrieval distribution; the same mechanism that closes catalog-misalignment gaps could in principle be used to bias retrieval toward a chosen subset of tools, and any practitioner reusing this method should be aware that the rewriter's behaviour is encoder-specific. We see no near-term dual-use concern beyond what already applies to any open dense retriever or instruction-tuned LLM.

Q.4 AI Assistant Use

For prototyping the codebase and experimentation, as well as for writing and editing of this manuscript, Claude Code with the Opus-4.5 model was used; all technical content, experimental design, claims, and figures are the authors' own.