CROSSMAX: CONTINUAL MODEL ROUTING WITH PREFERENCES AND ZERO-SHOT REGISTRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Public model hubs expand continually as new models are published, each requiring discoverability well before evaluation on any real query. Selecting the right model for a query in this setting faces two obstacles. First, users already express preferences in language—a preferred size, language, or publisher—yet routing methods typically ignore it, being trained only to satisfy the functional need, or apply that preference as a filter after scoring. Second, a router must select a model without executing it. Routers rely on past queries, but newly registered models have none. We propose CrossMax, a preference-guided router that supports zero-shot routing. CrossMax relies on two components: a prototype router that trains a small encoder on preference-augmented queries, letting stated preferences shape the ranking directly rather than filter it afterward, and a frozen embedding of the whole model card for models that have never been used. Each component outputs a winner and a margin rule compares the two by their model cards. The margin, set at inference time, chooses how readily a newly registered model is returned. Extending CMRBench, the reference continual routing benchmark, with preferences drawn from model cards, we show that training on the preference text makes the ranking follow the preference, while the same architecture trained without preferences and tested with them does not catch up even when paired with a parsed filter, and that a conservative margin keeps most of the accuracy on models the hub already knows while still returning models that have never been used.

1 INTRODUCTION

Public model hubs such as Hugging Face continue to grow, each new model arriving with a card long before anyone has used it (HuggingFace). A router for that hub still has to pick one model before any candidate is run. The pool is not a fixed catalogue: models keep appearing, so the router has to keep choosing well among the models the hub already knows while also being able to return models that have just been published. [1]

Users already say what they want: "I need a Google model", "I need a small model for image segmentation", "that works in Japanese". RouteLLM trains on which of the two models answered better, and the query states no such constraint (Ong et al., 2024). RouterBench evaluates routers over a fixed set of LLMs, again with no card constraint in the query (Hu et al., 2024). HuggingGPT filters the hub by task and ranks the rest by downloads (Shen et al., 2023). Arch-Router maps the query to a domain-action policy and a table names the model (Tran et al., 2025). ToolOrchestra writes a preference into the instruction and trains the orchestrator with a reward on which tools the trajectory called (Su et al., 2025). When a preference is honoured at all, it is applied after the ranking, by discarding models that do not satisfy it. This depends on parsing the preference correctly out of free text, and any part the parser misses is lost. We ask whether a ranker trained on the preference text can follow it directly.

Newly published models raise a different problem. Learned routers represent each model from the queries it has served: CARvE, the reference router of the CMRBench continual routing benchmark, learns one vector per model from its labelled queries (Bell et al., 2026). A newly published model has no such queries. Some routers use a new model by running it on probe prompts (Jitkrittum et al.,

[1] Our anonymous code is available here: Github

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.

2026; Cao et al., 2026), but on a hub where thousands of models arrive and the router must answer before any model is run, the model card is the one signal available from the moment of registration. A frozen text encoder can embed every card, so in principle cards alone could rank the whole pool. In practice, a representation learned from labelled queries is far stronger on the models it has seen: in our experiments, card embeddings alone pick the right known model less than half as often. The difficulty is therefore not scoring a new model but deciding when to prefer it, since its score comes from its card while a known model's score comes from a learned representation, and the two are not on the same scale. A newly published model often covers a task the hub already serves, so returning it instead of a proven older model should be a choice the router can tune.

In this paper, we propose CrossMax (Section 2), a router that keeps the two representations apart rather than mixing their scores (Figure 1). We call models with labelled queries *warm* and newly registered models *cold*. A prototype router, trained on queries that state a preference, selects the best warm model, and a frozen embedding of each model card selects the best cold model. The two winners are then compared through their card embeddings, and a margin, set at inference time without retraining, decides how often the cold winner is returned. The margin can be calibrated on earlier stages of the hub's growth.

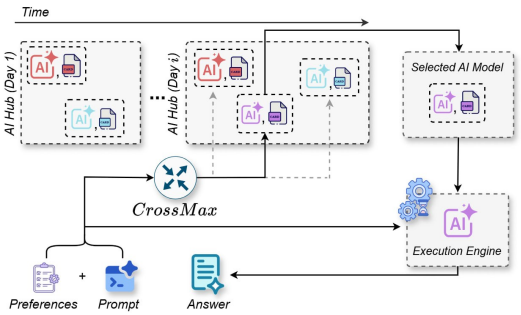


Figure 1: CrossMax on a growing hub. The prototype router picks the best warm model, the card embedder picks the best cold model, and a margin rule chooses which one to execute.

To evaluate preference following, we introduce CMRBench-Pref (Section 3). CMRBench (Bell et al., 2026) is a continual routing benchmark in which more than two thousand models and 34,239 requests arrive over four stages, drawn from API-calling and tool-use benchmarks and from Hugging Face, with a single correct (gold) model per request. CMRBench-Pref writes a natural-language preference taken from the gold model's card in front of every request, such as a publisher, a language, or the GPU the model must fit.

On the original CMRBench queries, the prototype router is the strongest continual router we compare (41.8% against 39.8% for CARvE) and forgets about half as much. On CMRBench-Pref, training on the preference reaches 46.9%, against 41.9% for the same router trained without preferences and paired with a parsed filter. When new models join the hub, a conservative margin raises accuracy over all queries from 33.1% to 35.7%, while ranking the full pool with a single score never improves on returning only warm models.

In summary, our contributions are:

1. **CrossMax**, a router for growing hubs that selects a warm winner with a prototype router trained on preference-augmented queries and a cold winner with a frozen embedding of each model card, and chooses between them with a margin set at inference, so that a model can be returned from the moment it is registered, without retraining.
2. **CMRBench-Pref**, which adds a natural-language preference over five card fields (task, GPU, publisher, language, and packaging) to each of the 34,239 requests of the CMRBench continual routing benchmark, together with a protocol that registers the models of a later stage as never used.
3. **Evidence that the two components are complementary** (Section 3): the prototype router follows preferences but cannot return a new model, the card embedding can, and a margin between them beats single-score rankings of the full pool.

2 CrossMax: Choosing Between Two Representations

CrossMax has two components and a decision rule. A *prototype router* ranks warm models by their class means (Section 2.1), and a frozen *card embedder* ranks cold models by their cards (Section 2.2). A *margin rule* then returns one of the two winners (Section 2.3).

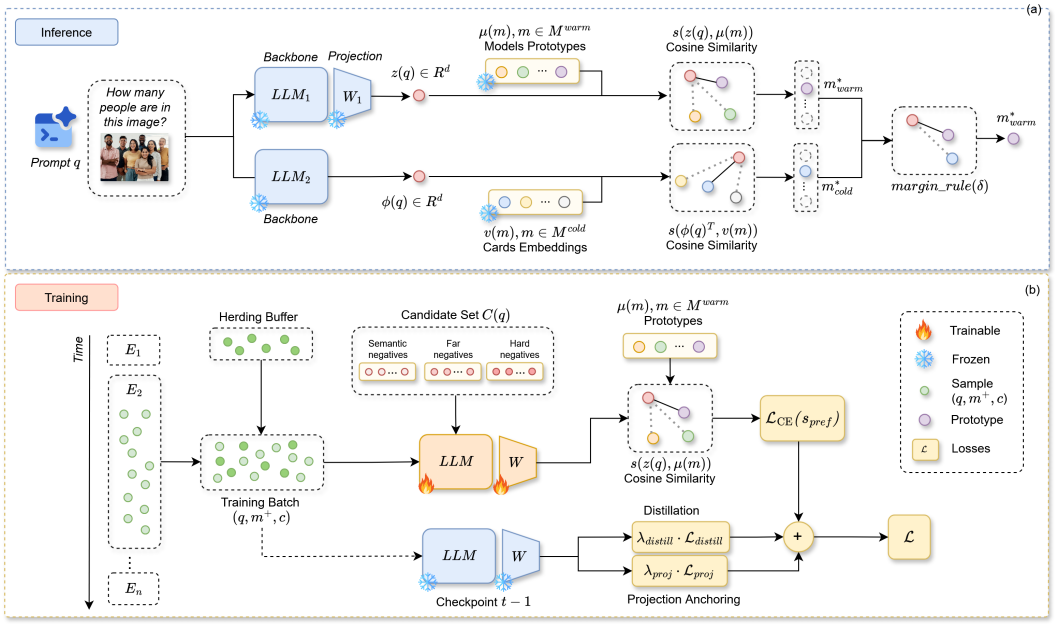


Figure 2: Overview of CrossMax. **(a) Inference.** The prototype router scores the query against the class means of warm models, and the frozen card embedder scores it against the card embeddings of cold models. Each returns a winner, and the margin rule returns the cold winner only if its card is closer to the query by more than δ. **(b) Training.** Only the prototype router is trained, on current and replayed queries that state a preference, with a cross-entropy over sampled candidates whose scores include a preference penalty, plus distillation and a projection anchor to the previous checkpoint.

Setting and notation. Models and queries arrive over four experiences in a class incremental stream $\mathcal{E}_1, \ldots, \mathcal{E}_T$ as defined in Bell et al. (2026). After experience t the hub contains the models $\mathcal{M}_{\le t}$. Each query q has one gold model $m^\star(q)$, and the router must return one identifier from $\mathcal{M}_{\le t}$ before any model is run.

Warm and cold models. A model is *warm* once it has at least one labelled query. The warm group $\mathcal{M}_t^{\text{warm}}$ contains these models, and each has a class mean $\mu(m)$ computed from its labelled queries. The cold group $\mathcal{M}_t^{\text{cold}} = \mathcal{M}_{\le t} \setminus \mathcal{M}_t^{\text{warm}}$ contains models that have a card but no labelled query; each has only a card vector $v(m)$. A query is warm or cold according to its gold model.

Card fields. Each model card is summarised by a record over five fields, $f \in \{\text{task}, \text{GPU}, \text{publisher}, \text{language}, \text{packaging}\}$, and m_f denotes the value of field f on the card of m. Task is the Hub pipeline tag. GPU is the smallest standard memory tier that the model's estimated inference footprint fits (Appendix F).

Preferences. A preference is a partial record c: some fields are filled and the rest are empty. In CMRBench-Pref, c is sampled from the gold card and written as a natural-language *preference prefix* in front of the request (Appendix F). During training the router receives the gold preference c alongside the text. At evaluation it sees only the text and must recover an estimate $\hat{c}$ from it (Appendix G). The *feasible set* $F(c) \subseteq \mathcal{M}_{\le t}$ contains the models whose cards match every filled field of c, so $m^\star(q) \in F(c)$ by construction. Sat@k is the fraction of the top-k returned models that lie in $F(c)$.

2.1 Prototype Router

The prototype router is the only trained component. It scores each warm model against a prototype built from stored query states rather than a learned per-model vector, so we call it a prototype NCM router.

Query encoder. The encoder is Qwen3-Embedding-0.6B (Zhang et al., 2025a) with a trainable LoRA on the frozen base (Hu et al., 2022), followed by a bias-free linear map W. Writing $h(q)$ for the pooled encoder state, the query vector is

$$z(q) = \text{normalize}\big(Wh(q)\big). \tag{1}$$

The query is the full request text, preference prefix included.

Prototypes. Each warm model m keeps the pooled states $h(q)$ of its $n(m)$ labelled training queries. These states are not re-encoded by later encoders, apart from one training-time overwrite (Appendix C.1). At every scoring pass they are mapped through the current W, and the prototype $\mu(m)$ is their normalised mean, shrunk toward a taxonomy ancestor of m in proportion to how few queries it rests on (Appendix C.2). Prototypes are detached and receive no gradient, but $\mu(m)$ moves as W trains. Stored states let an old model's prototype rest on its labelled queries rather than on the zero or one query that a 10% buffer holds for it - Table 5 measures what this contributes.

The router scores a warm model by the scaled cosine

$$s(q, m) = \frac{z(q)^\top \mu(m)}{\tau}, \qquad \tau = 0.08. \tag{2}$$

Preference penalty. Each value a field can take, such as *Google* or *12GB*, has a learned 16-dimensional vector; C_f is the table of vectors for field f. A preference and a model are embedded over the fields the preference fills,

$$p(c) = \sum_{f\,:\,c_f \neq \varnothing} C_f[c_f], \qquad p(m) = \sum_{f\,:\,c_f \neq \varnothing} C_f[m_f], \tag{3}$$

and the full score subtracts their squared distance,

$$s_{\text{pref}}(q, m) = s(q, m) - a\,\big\|p(c) - p(m)\big\|^2, \tag{4}$$

where a is a learned scalar. Training uses the gold preference c; evaluation uses the extracted $\hat{c}$ (Appendix G).

Objective. The loss is a cross-entropy of s_{pref} over $K = 512$ sampled candidates, plus two continual regularisers: a distillation term matching h on replayed queries to a snapshot of the previous encoder, and an anchor pulling W toward its previous value. Replay queries are chosen by herding within a fixed budget. Class means receive no gradient. The full objective is in Appendix D.

2.2 Card embedder

The card embedder ϕ is a frozen Qwen3-Embedding-8B. At registration, each model's card is rendered as a document $d(m)$ containing its catalogue fields and full description (Appendix B), and its card vector is

$$v(m) = \text{normalize}\big(\phi(d(m))\big). \tag{5}$$

A query is scored against a cold model by $\phi(q)^\top v(m)$. The card embedder can be replaced by a smaller one without retraining; larger embedders benefit from richer cards (Appendix B, Table 6).

2.3 Margin rule

When both groups are occupied. Each scorer names a winner in its own group:

$$m^\star_{\text{warm}}(q) = \operatorname*{arg\,max}_{m \in \mathcal{M}_t^{\text{warm}}} s(q, m), \qquad m^\star_{\text{cold}}(q) = \operatorname*{arg\,max}_{m \in S(q)} \phi(q)^\top v(m), \tag{6}$$

where $S(q) = \mathcal{M}_t^{\text{cold}} \cap F(\hat{c})$ if that set is non-empty and $\mathcal{M}_t^{\text{cold}}$ otherwise. The warm winner uses the plain cosine: the encoder was trained on preference-bearing queries, so $z(q)$ already carries the preference as a soft constraint. The frozen card embedder was never trained on preferences and cannot, so the cold winner is restricted to $F(\hat{c})$ instead. Because z and ϕ live in different spaces, their scores are not comparable, so the rule compares the two winners using ϕ alone. It returns the cold winner only if

$$\phi(q)^\top v\big(m^\star_{\text{cold}}(q)\big) > \phi(q)^\top v\big(m^\star_{\text{warm}}(q)\big) + \delta, \tag{7}$$

Table 1: CMRBench-Pref stream. New M counts models first seen in each experience. k is the number of preference fields stated, and $|F(c)|$ is the feasible-set size, both on the eval split.

	Stream	Train	Eval	New M	k=1	k=2	k=3	Med. $\lvert F(c)\rvert$
$\mathcal{E}_1$	APIBench	7826	867	852	233	575	59	45
$\mathcal{E}_2$	MLLM	4582	803	359	127	595	81	67
$\mathcal{E}_3$	HB-1	8847	984	450	192	749	43	39
$\mathcal{E}_4$	HB-2	9297	1033	464	201	793	39	31
All		30,552	3687	2125	753	2712	222	44

and the warm winner otherwise, including on ties. A large δ keeps the warm model unless the cold card is clearly closer; a small δ returns the cold model whenever its card is at all closer.

When every model is warm. Once $\mathcal{M}_t^{\text{cold}}$ is empty, the card embedder is not run and CrossMax reduces to the prototype router: it returns $\arg\max_{m \in \mathcal{M}_t^{\text{warm}}} s_{\text{pref}}(q, m)$ with $\hat{c}$ from the query and no mask after scoring. This is the setting of all headline results in Table 3.

3 Experiments

3.1 Benchmarks and protocols

Our experiments address three questions. First, does a ranker trained on queries that state a preference follow that preference, or does it still rely on a filter applied after scoring (Section 3.2)? Second, is the prototype router a strong continual router on CMRBench and on CMRBench-Pref (Section 3.3)? Third, once cold models can also be scored, how much warm accuracy must be traded for cold accuracy (Section 3.4)? We trace this trade-off by varying the margin δ of the margin rule (Section 2.3).

CMRBench-Pref. CMRBench-Pref extends CMRBench by writing a preference prefix in front of each request. Everything else is unchanged: the same four experiences in temporal order (APIBench, then MLLM, then two Hugging Face slices (Patil et al., 2024; Wang et al., 2025; Bell et al., 2026)), the same model pool, and the same single gold model per query (Table 1; construction details in Appendix F). Task, publisher, and language preferences are taken directly from the gold model's card. Size preferences are stated as a GPU, and a model satisfies one when its estimated inference footprint fits that GPU's memory tier (Appendix F). The same tier is sometimes phrased loosely, as "one GPU" or "a consumer GPU", so the prefix does not simply copy a tier into the query. Every evaluation query states at least one preference and most state two, which narrows the feasible set $F(c)$ to a median of 44 models.

Protocol. Headline results in Table 3 are measured after $\mathcal{E}_4$ as the mean over all cells on or below the diagonal of the 4×4 experience matrix, one cell for each pair (t, j) with $t \geq j$ (Rodríguez et al., 2018). Trained rows report mean $\pm$ standard error over seeds 40–42; retrieval baselines use seed 40 and are marked †. Joint rows are trained on all four experiences at once and report the mean accuracy of that single checkpoint over the four evaluation sets.

The prototype router encodes the full query, preference prefix included. At evaluation, the preference penalty of Eq. 4 uses the extracted preference $\hat{c}$ recovered from that same query (Appendix G), and no hard mask is applied after scoring. CARvE and the retrieval baselines have no preference penalty and rank by cosine alone.

The margin rule is evaluated separately, after $\mathcal{E}_3$. We register the 464 models first seen in $\mathcal{E}_4$ as cold, score them with the card embedder on the whole card (Section 2.2), and apply Eq. 7. The margin rule is not trained. It is an inference-time rule that chooses between the warm and cold winners, and δ is calibrated on an earlier stage of the stream: the $\mathcal{E}_2$ checkpoint with the $\mathcal{E}_3$ models registered as cold (Appendix H).

Metrics. Model accuracy (M-Acc) asks whether the returned model is the gold model. Domain accuracy (D-Acc) is coarser: a prediction counts as correct when the returned model shares a domain with the gold model. On CMRBench-Pref the preference prefix often names the task, so domain ac-

Table 2: Does the prototype router follow the stated preference? Drop prefix removes the preference from the query. Swap GPU changes the stated GPU to one the gold model does not fit, so a lower M-Acc means stronger following.

Slice	n	Med. $\lvert F(c)\rvert$	M-Acc	Sat@1	Sat@5	Sat@10
All	3687	44	46.9 ± 0.3	96.6 ± 0.1	84.7 ± 0.2	76.2 ± 0.3
k=1	753	49	45.9 ± 0.8	96.7 ± 0.5	87.2 ± 0.5	80.5 ± 0.6
k=2	2712	44	45.3 ± 0.3	96.9 ± 0.0	86.2 ± 0.1	78.0 ± 0.3
k=3	222	6	61.0 ± 0.2	93.9 ± 0.3	62.4 ± 0.1	46.0 ± 0.2
Has task	3332	44	46.4 ± 0.3	96.6 ± 0.1	84.7 ± 0.2	76.2 ± 0.4
No task	355	84	50.2 ± 0.9	97.0 ± 0.3	84.3 ± 0.2	75.8 ± 0.2
Has GPU	2671	45	46.4 ± 0.3	96.8 ± 0.0	85.7 ± 0.1	77.9 ± 0.3
Has publisher	576	13	48.9 ± 0.5	97.0 ± 0.2	82.6 ± 0.2	69.3 ± 0.3
Has language	249	12	66.0 ± 0.9	91.5 ± 0.6	54.1 ± 0.0	40.3 ± 0.2
Drop prefix	3687	44	32.9 ± 0.3	57.5 ± 0.2	45.7 ± 0.3	40.3 ± 0.4
Swap GPU	2671	27	11.8 ± 0.9	37.6 ± 4.0	35.7 ± 3.5	33.0 ± 3.4

curacy can largely be read off the query and is not a substitute for model accuracy. Sat@1 (Section 2) is the fraction of queries whose top-ranked model lies in $F(c)$; on those queries, a filter applied after scoring would not change the top-ranked model. Forgetting (M-Fgt, D-Fgt) is the drop, by the end of $\mathcal{E}_4$, from the accuracy measured just after each earlier experience was learned (Lopez-Paz & Ranzato, 2017). For the margin rule we report three accuracies on query subsets rather than on the experience matrix. Warm accuracy is model accuracy on warm queries (n=2767), cold accuracy is model accuracy on cold queries (n=920), and mixed accuracy is model accuracy on both together (n=3687). Full definitions are in Appendix E.

3.2 Does training on preferences beat filtering?

Trained on the preference prefix, the ranking follows it. The prototype router trained on CMRBench-Pref with 10% herding replay reaches 46.9% model accuracy (Table 3), and its top-ranked model lies in $F(c)$ on 96.6% of queries, so a filter applied after scoring would change the top-ranked model on fewer than 4% of them. Training is what makes the prefix useful. Frozen Qwen3-Embedding-0.6B sees the same query text without any training and reaches Sat@1 48.2 and M-Acc 10.6; BM25 reaches Sat@1 25.3 and M-Acc 4.3. Both baselines are scored on the same evaluation queries as the 46.9, so writing the preference into the query is not enough on its own.

A parsed filter does not substitute for training on the preference. We train the same prototype router on the original CMRBench requests and score it on the CMRBench-Pref queries (Table 3, bottom). Left in the query, the prefix is text this router never saw, and accuracy falls to 34.3. The fair comparison strips the prefix first: the router then reaches 42.2, and masking its ranking with a regex parse of the preference raises Sat@1 from 75.3 to 84.8 but leaves accuracy at 41.9. An oracle mask to the true F(c) lifts the stripped ranking to 52.2, above every trained variant, so recovering the preference accurately matters more than training on it.

The trained encoder reads the preference prefix. Table 2 probes the CMRBench-Pref model from the first paragraph. Removing the preference prefix from its test queries lowers M-Acc from 46.9 to 32.9 and Sat@1 from 96.6 to 57.5. Swapping the stated GPU tier to one the gold model fails, on queries where another model meets the new tier, moves the top-ranked model to a model in the new feasible set on 37.6% of queries, and the router returns the now-infeasible gold model on only 11.8%. Here a lower M-Acc means the router is following the preference. Sat@1 stays at or above 96% when the query states a single field (96.7) and when it names no task (97.0), so the 46.9 does not come from a task label copied from the input.

3.3 Is the router a strong continual learner?

The prototype router is the strongest continual router we compare. On CMRBench-Pref at 10% replay, it reaches 46.9 model accuracy against 43.5 for CARvE, with lower forgetting (12.5 against 21.3). On the original CMRBench requests, which state no preference, the comparison is 41.8

Table 3: Continual routing after $\mathcal{E}_4$: accuracy over 3 seeds († one deterministic seed; ‡ seed 40 only). CMRBench-Pref prototype-router rows use the preference penalty with extracted $\hat{c}$. *Prefix stripped* removes the preference before ranking; filters apply after ranking. Bold: best continual 0.6B result.

	Accuracy (%) (↑)			Forgetting (%) (↓)	
Setting	**M-Acc**	**D-Acc**	**Sat@1**	**M-Fgt**	**D-Fgt**
Trained and evaluated on CMRBench-Pref					
Retrieval					
BM25† (Robertson & Zaragoza, 2009)	4.3	26.8	25.3	–	–
Frozen Qwen3-0.6B† (Zhang et al., 2025a)	10.6	55.4	48.2	–	–
Continual					
Prototype router, 10% replay	46.9 ± 0.3	93.4 ± 0.2	96.6 ± 0.1	$\mathbf{12.5 \pm 1.1}$	2.3 ± 0.3
5% replay	$\mathbf{47.5 \pm 0.3}$	$\mathbf{93.4 \pm 0.1}$	$\mathbf{96.8 \pm 0.1}$	12.9 ± 0.6	$\mathbf{1.8 \pm 0.0}$
2% replay	46.4 ± 0.6	93.0 ± 0.3	96.5 ± 0.1	17.9 ± 0.5	3.1 ± 0.5
1% replay	44.7 ± 0.2	92.3 ± 0.1	96.0 ± 0.1	22.6 ± 1.0	4.6 ± 0.2
CARvE, 10% replay (Bell et al., 2026)	43.5 ± 0.1	91.8 ± 0.2	93.1 ± 0.5	21.3 ± 1.0	4.9 ± 0.1
5% replay	37.9 ± 0.2	89.3 ± 0.3	89.9 ± 0.3	30.7 ± 0.5	8.6 ± 0.5
2% replay	32.1 ± 0.3	85.9 ± 0.3	84.8 ± 0.7	42.3 ± 0.6	15.8 ± 0.2
1% replay	29.7 ± 0.1	83.8 ± 0.5	80.9 ± 1.1	46.3 ± 0.6	19.1 ± 1.2
Scale (Qwen3-Emb-8B encoder)					
Prototype router, 10% replay	48.8 ± 0.1	94.1 ± 0.1	97.0 ± 0.0	11.9 ± 0.3	1.9 ± 0.2
Upper bound (non-continual)					
Joint prototype router	50.9 ± 0.2	92.8 ± 0.0	96.3 ± 0.2	–	–
Joint CARvE	53.6 ± 0.6	94.4 ± 0.4	96.5 ± 0.5	–	–
Trained and evaluated on CMRBench					
BM25†	4.3	16.7	–	–	–
Prototype router, 10% replay	41.8 ± 0.3	80.9 ± 0.1	–	10.5 ± 0.1	6.5 ± 0.0
CARvE, 10% replay	39.8 ± 0.4	78.9 ± 0.3	–	20.1 ± 0.2	11.8 ± 0.3
Trained on CMRBench, evaluated on CMRBench-Pref					
Prototype router, 10% replay	34.3 ± 0.3	79.5 ± 0.5	76.0 ± 0.2	–	–
+ parse-then-filter	31.7 ± 0.1	78.1 ± 0.2	80.8 ± 0.2	–	–
+ oracle filter	39.9 ± 0.2	–	–	–	–
Prefix stripped from eval‡	42.2	80.7	75.3	–	–
+ parse-then-filter‡	41.9	83.6	84.8	–	–
+ oracle filter‡	52.2	95.4	100	–	–

against 39.8, with forgetting of 10.5 against 20.1. After $\mathcal{E}_4$ the cold group is empty, so CrossMax reduces to the prototype router and both comparisons are between warm rankers. CARvE overtakes it only under joint training (53.6 against 50.9), an upper bound rather than a continual result. The Qwen3-Embedding-8B encoder adds 1.9 points (48.8), so the 0.6B encoder gives up little.

Prototype router holds with little replay. At 5% replay the prototype router matches its 10% result (47.5 against 46.9, within seed noise). At 1% its model accuracy falls to 44.7, while Sat@1 stays at 96.0 and forgetting rises to 22.6. CARvE at the same 1% budget falls to 29.7, with forgetting of 46.3, and the gap widens at every step down in replay. The prototype router's class means are rebuilt from stored query states, so the buffer only has to keep the encoder from drifting, with help from distillation and the projection anchor; CARvE's per-model rows are learned, so replay is what holds them in place (Table 5). The lead does not depend on keeping every query's state: rebuilding the class mean of every replayed model from its replay text, which leaves 672 of the $\mathcal{E}_1$ models scored from a single exemplar at 10% replay, changes accuracy by less than one point at every budget (seed 40, Table 5), and the router stays above CARvE at the same replay fraction.

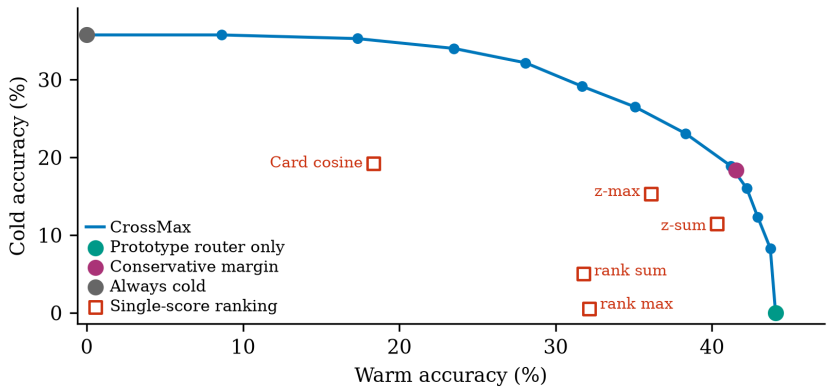


Figure 3: Warm vs. cold accuracy after E3, with the E4 models registered as cold. The curve sweeps the margin δ; squares rank the full pool with a single score.

3.4 Returning cold models

Returning a cold model costs warm accuracy. We evaluate the margin rule after $\mathcal{E}_3$, with the 464 models first seen in $\mathcal{E}_4$ registered as cold (Section 3.1). The warm winner is the class-mean cosine of Eq. 2: the encoder was trained on preference-bearing queries, so the preference already acts as a soft constraint in $z(q)$. The cold winner is the model in the extracted feasible set $F(\hat{c})$ whose whole card is closest under the frozen 8B card embedder; if that set is empty, the rule falls back to the unmasked cosine (Appendix G). The frozen embedder was never trained on preferences, and this explicit mask is what makes it follow them: always returning the cold winner reaches 35.8 cold accuracy with the mask and 27.5 without it (Table 4).

Returning only the warm winner gives 44.1 warm accuracy and, by construction, 0 cold accuracy. This differs from the headline 46.9 in three ways: it is measured after $\mathcal{E}_3$ rather than $\mathcal{E}_4$, on query subsets rather than the experience matrix, and without the preference penalty. Sweeping δ traces the trade-off between the two groups, from never returning the cold winner to always returning it (Figure 3, Table 4).

A conservative margin keeps most of the warm accuracy. We calibrate the conservative margin on an earlier stage of the stream, as the δ that returns the cold winner on 20% of $\mathcal{E}_3$ queries at the $\mathcal{E}_2$ checkpoint (Appendix H). Applied unchanged here, it returns the cold winner on 18.8% of queries, keeps 41.5 warm accuracy against 44.1, and reaches 18.4 cold accuracy. Mixed accuracy rises from 33.1 to 35.7, because some cold queries are now answered correctly while most warm queries keep their answer. Always returning the cold winner raises cold accuracy to 35.8 and drops warm accuracy to 0.

Ranking the full pool with one score does not do better. Instead of choosing between two winners, one can rank all 2125 models with a single score (Table 4, lower block). Card cosine alone reaches 18.4 warm and 19.2 cold accuracy. Combining the prototype-router and card scores across the pool trades one group against the other without reaching the CrossMax curve: z-score max gives 36.1 warm and 15.3 cold, z-score sum 40.3 and 11.5, and rank-based combinations return almost no correct cold model (0.5 and 5.1). None of these exceeds the 33.1 mixed accuracy of never returning a cold model, whereas the conservative margin reaches 35.7. While cold models exist, each query needs one extra 8B encode, 15.3× the cost of a 0.6B encode, and training cost is unchanged (Appendix H, Figure 4).

4 Related work

The introduction reviews how routers treat stated preferences: most train only on the functional request, and the rest apply the preference as a filter or a policy table. CMRBench grows its pool across experiences but keeps queries as short task requests, so preference language never enters training (Bell et al., 2026). CMRBench-Pref puts the preference in the query so the encoder can be

trained on that text, which is a different intervention from receiving the same constraint only as a mask after the ranking has already been made.

Where the pool grows, a new model is typically represented from a probe. UniRoute embeds a new LLM by prediction-error vectors on representative prompts (Jitkrittum et al., 2026), Model-SAT admits a model by aptitude tests (Zhang et al., 2025b), SCOPE fingerprints a new LLM by answers on an anchor bank (Cao et al., 2026), and Model Spider uses transferability estimators as ranking supervision (Zhang et al., 2023). RouteProfile builds a graph from public signals in cards and reports instead of a probe, but over a small pool of foundation models (Xu et al., 2026). Cascades and post-inference ensembles execute candidates before deciding (Chen et al., 2024; Jiang et al., 2023; Guha et al., 2024), and tool-use benchmarks and API-calling models such as Gorilla select among APIs rather than hub models (Patil et al., 2024; Li et al., 2023; Xu et al., 2023; Qin et al., 2023). A hub of thousands of those models does not receive such probes at publication, and the router must return an identifier before the new model is run. The card exists from the start, so its embedding can score the model with no labelled query, leaving open when that model should be returned.

Once labelled queries exist, a class mean is the ordinary prototype of class-incremental learning, with model identifiers as classes (van de Ven et al., 2022; Parisi et al., 2019; Kirkpatrick et al., 2017). Prototype methods estimate each class from labelled examples on a capable representation (Zhou et al., 2025; McDonnell et al., 2023; Goswami et al., 2023; Rebuffi et al., 2017), as our prototype router does, and few-shot and long-tailed variants still require at least one example before a prototype of that class exists (Zhou et al., 2022; Zou et al., 2024; Qi et al., 2025). Before any example, side-information constructions already write a class from attributes, as a cascade of attribute classifiers (Lampert et al., 2014), a language embedding of the class name (Frome et al., 2013), or a class vector formed from attributes (Akata et al., 2013), and those methods score an image against such a vector, which is a different object from a routing representation built from the card of a growing hub. Prior work treats the two constructions as alternative ways to represent a class, yet a card embedding used on its own is weaker than the class mean (Section 3.4), which is why it must not replace it. They occupy two cells of the same hub—prototypes after traffic, card embeddings when a model first appears. CrossMax keeps both: each component names a winner in its own group, and the choice between those winners is made from their cards, by a margin set at inference.

5 Discussion and conclusion

We introduced CMRBench-Pref, which writes a stated preference into the query on a growing hub, and CrossMax, which compares a warm winner from the prototype router with a cold winner from the card embedder. Training on preference-bearing queries raises accuracy over the same router with a parsed filter, while the preference penalty is what makes the top-ranked model satisfy the preference; an oracle filter shows how much better parsing could do. A newly published model has no labelled queries but can still be scored from its card. The card embedding alone is weaker than the class mean (18.4 against 44.1 warm accuracy), so it complements rather than replaces it. A margin set from earlier experience decides how readily the cold winner is returned: a conservative setting keeps most warm accuracy and still returns never-used models, while single-score rankings of the whole pool do not exceed the mixed accuracy of never returning a cold model.

Limitations. CMRBench-Pref writes the preference as a prefix sampled from the gold card. That is the right protocol for asking whether a ranking can follow constraints that were stated, and it is not a log of what users type. Returning a model that has never been used is the point of the cold group; because the model is chosen before it has been run, a hub that accepts the cold winner should watch it at execution. The oracle filter (52.2) shows that errors in the extracted preference $\hat{c}$ remain the largest source of lost accuracy, so a stronger extractor may matter more than a stronger ranker. The margin is calibrated on a single earlier stage, and a hub whose arrival rate or domain mix shifts sharply may need it recalibrated.

Future work. A natural continuation lets CrossMax propose a small candidate set that is then executed and scored, with the preference shaping both stages. Testing that requires prompts and constraints people actually write, rather than preferences sampled from cards.

AI USE STATEMENT

In this work, we used generative AI tools to help design and provide feedback on research methodology and experiments, to assist in interpreting results, to help implement methods and to generate synthetic datasets. We have not used generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, assist with translation, clean or reformat datasets, or support qualitative and thematic data analysis. Additionally, we used generative AI tools for brainstorming and for editing the manuscript to improve readability. We have reviewed all AI-assisted work. AI-assisted feedback on experimental design was evaluated and adopted or rejected by the authors before any experiments were run; AI-assisted interpretation of results was checked against the underlying tables and figures by the authors prior to inclusion in the text; AI-assisted code (including the adaptation, routing, and evaluation harness) was reviewed and tested for correctness by the authors before being used to produce any reported result; and AI-assisted manuscript edits were reviewed by the authors for technical accuracy. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses CMRBench (Bell et al., 2026), an existing public benchmark. It contains model identifiers, task descriptions, and public model-card text, not personal data.

One consideration is specific to the method: cold-start routing sends users to a model that has never been executed or scored on real queries. This could route traffic to a newly registered model with unknown safety, licensing, or quality issues that only appear at execution time. This risk exists for any pre-inference router, but it is sharper here, since CrossMax is designed to route to models with no track record. We recommend pairing deployment with execution-time monitoring rather than treating pre-inference confidence alone as sufficient.

REPRODUCIBILITY STATEMENT

We release all code in an anonymous codebase, including the training pipeline, evaluation scripts and data curation code.

REFERENCES

Zeynep Akata, Florent Perronnin, Zaïd Harchaoui, and Cordelia Schmid. Label-embedding for attribute-based classification. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pp. 819–826. IEEE Computer Society, 2013. doi: 10.1109/CVPR.2013.111. URL https://doi.org/10.1109/CVPR.2013.111.

Jack Bell, Giacomo Carfi', Gerlando Gramaglia, and Vincenzo Lomonaco. Continual model routing in evolving model hubs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=IT7lA8t0S6.

Qi Cao, Shuhao Zhang, Ruizhe Zhou, Ruiyi Zhang, Peijia Qin, and Pengtao Xie. Models under SCOPE: Scalable and controllable routing via pre-hoc reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ydQkpswDPi.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=cSimKw5p6R.

Andrea Frome, Gregory S. Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomás Mikolov. Devise: A deep visual-semantic embedding model. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2121–2129, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/7cce53cf90577442771720a370c3c723-Abstract.html.

Dipam Goswami, Yuyang Liu, Bartł omiej Twardowski, and Joost van de Weijer. Fecam: Exploiting the heterogeneity of class distributions in exemplar-free continual learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 6582–6595. Curran Associates, Inc., 2023. doi: 10.52202/075280-0288. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/15294ba2dcfb4521274f7aa1c26f4dd4-Paper-Conference.pdf.

Neel Guha, Mayee F. Chen, Trevor Chow, Ishan S. Khare, and Christopher Re. Smoothie: Label Free Language Model Routing. November 2024. URL https://openreview.net/forum?id=pPSWHsgqRp.

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, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A Benchmark for Multi-LLM Routing System. July 2024. URL https://openreview.net/forum?id=IVXmV8Uxwh.

HuggingFace. Hugging Face – The AI community building the future. URL https://huggingface.co/.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient LLM inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ka82fvJ5f1.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017. doi: 10.1073/pnas.1611835114. URL https://www.pnas.org/doi/10.1073/pnas.1611835114.

Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(3):453–465, 2014. doi: 10.1109/TPAMI.2013.140. URL https://doi.org/10.1109/TPAMI.2013.140.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187.

Zhizhong Li and Derek Hoiem. Learning without Forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(12):2935–2947, December 2018. ISSN 0162-8828. doi: 10.1109/TPAMI.2017.2773081. URL https://doi.org/10.1109/TPAMI.2017.2773081.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6470–6479, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Mark D. McDonnell, Dong Gong, Amin Parvaneh, Ehsan Abbasnejad, and Anton van den Hengel. Ranpac: Random projections and pre-trained models for continual learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 12022–12053. Curran Associates, Inc., 2023. doi: 10.52202/075280-0526. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/2793dc35e14003dd367684d93d236847-Paper-Conference.pdf.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to Route LLMs from Preference Data. October 2024. URL https://openreview.net/forum?id=8sSqNntaMr.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, May 2019. ISSN 0893-6080. doi: 10.1016/j.neunet.2019.01.012. URL https://www.sciencedirect.com/science/article/pii/S0893608019300231.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. *Advances in Neural Information Processing Systems*, 37:126544–126565, December 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Zhi-Hong Qi, Da-Wei Zhou, Yiran Yao, Han-Jia Ye, and De-Chuan Zhan. Adaptive adapter routing for long-tailed class-incremental learning. *Mach. Learn.*, 114(3):68, 2025. doi: 10.1007/S10994-024-06662-4. URL https://doi.org/10.1007/s10994-024-06662-4.

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, October 2023. URL http://arxiv.org/abs/2307.16789.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5533–5542. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.587. URL https://doi.org/10.1109/CVPR.2017.587.

Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009. ISSN 1554-0669, 1554-0677. doi: 10.1561/1500000019. URL http://www.nowpublishers.com/article/Details/INR-019.

Natalia Díaz Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don't forget, there is more than forgetting: new metrics for Continual Learning. *CoRR*, abs/1810.13166, 2018. URL http://arxiv.org/abs/1810.13166. arXiv: 1810.13166.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/77c33e6a367922d003ff102ffb92b658-Abstract-Conference.html.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. ToolOrchestra: Elevating Intelligence via Efficient Model and Tool Orchestration, November 2025. URL https://arxiv.org/abs/2511.21689v1.

Co Tran, Salman Paracha, Adil Hafeez, and Shuguang Chen. Arch-Router: Aligning LLM Routing with Human Preferences, June 2025. URL http://arxiv.org/abs/2506.16655.

Gido M. van de Ven, Tinne Tuytelaars, and Andreas S. Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, December 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00568-3. URL https://www.nature.com/articles/s42256-022-00568-3.

Chenyu Wang, Weixin Luo, Sixun Dong, Xiaohua Xuan, Zhengxin Li, Lin Ma, and Shenghua Gao. MLLM-Tool: A Multimodal Large Language Model for Tool Agent Learning. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 6678–6687, February 2025. doi: 10.1109/WACV61041.2025.00650. URL https://ieeexplore.ieee.org/abstract/document/10943671.

Jingjun Xu, Hongji Pu, Tao Feng, Haozhen Zhang, Jiaxuan You, and Ge Liu. Routeprofile: Graph-based profiling for cold-start llm routing. *arXiv preprint arXiv:2605.00180*, 2026.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the Tool Manipulation Capability of Open-source Large Language Models, May 2023. URL http://arxiv.org/abs/2305.16504.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *CoRR*, abs/2506.05176, 2025a. doi: 10.48550/ARXIV.2506.05176. URL https://doi.org/10.48550/arXiv.2506.05176.

Yi-Kai Zhang, Ting-Ji Huang, Yao-Xiang Ding, De-Chuan Zhan, and Han-Jia Ye. Model Spider: Learning to Rank Pre-Trained Models Efficiently. *Advances in Neural Information Processing Systems*, 36:13692–13719, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/2c71b14637802ed08eaa3cf50342b2b9-Abstract-Conference.html.

Yi-Kai Zhang, De-Chuan Zhan, and Han-Jia Ye. Capability instruction tuning. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 25958–25966. AAAI Press, 2025b. doi: 10.1609/AAAI.V39I24.34790. URL https://doi.org/10.1609/aaai.v39i24.34790.

Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, Liang Ma, Shiliang Pu, and De-Chuan Zhan. Forward compatible few-shot class-incremental learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pp. 9036–9046. IEEE, 2022. doi: 10.1109/CVPR52688.2022.00884. URL https://doi.org/10.1109/CVPR52688.2022.00884.

Da-Wei Zhou, Zi-Wen Cai, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Revisiting class-incremental learning with pre-trained models: Generalizability and adaptivity are all you need. *Int. J. Comput. Vis.*, 133(3):1012–1032, 2025. doi: 10.1007/S11263-024-02218-0. URL https://doi.org/10.1007/s11263-024-02218-0.

Yixiong Zou, Shanghang Zhang, Haichen Zhou, Yuhua Li, and Ruixuan Li. Compositional few-shot class-incremental learning. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62964–62977. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/zou24c.html.

APPENDIX CONTENTS

A Additional Experiments **14**

B Card document **14**

C Protocol details **15**

C.1 Prototype lifecycle 16

C.2 Taxonomy ancestor 17

D Training losses **17**

E Metric definitions **17**

F CMRBench-Pref construction **18**

G Preference penalty and extracted $\hat{c}$ **18**

H The margin and the second encode **19**

A Additional Experiments

Table 4: Numbers behind Figure 3. Warm n=2767; cold n=920; mixed n=3687. Before mask, the cold winner is the best cold model by card cosine; after mask, it is the best cold model inside $F(\hat{c})$, which gives the curve. The conservative row uses one δ, calibrated at the $\mathcal{E}_2$ checkpoint with the $\mathcal{E}_3$ models registered as cold (Appendix H) and applied unchanged; after the mask it returns the cold winner on 18.8% of queries, so it is not the 20% point of the sweep. The 50% row is the midpoint of the sweep. After-mask cells and the lower block are mean ± SE over seeds 40–42; before-mask cells are means over the same seeds, as that ablation was not stored per seed. The lower block ranks all 2125 models with a single score, so it has no before-mask variant. Card-cosine standard errors are zero because the card embedder is frozen and shared across seeds.

	Before mask			After mask		
Policy	**Warm**	**Cold**	**Mixed**	**Warm**	**Cold**	**Mixed**
Prototype router only (δ large)	44.1	0.0	33.1	44.1 ± 0.3	0.0 ± 0.0	33.1 ± 0.2
Conservative margin	39.8	16.7	34.1	41.5 ± 0.3	18.4 ± 0.3	35.7 ± 0.1
50% cold	31.6	23.0	29.5	31.7 ± 0.3	29.2 ± 0.1	31.1 ± 0.2
Always cold (δ small)	0.0	27.5	6.9	0.0 ± 0.0	35.8 ± 0.1	8.9 ± 0.0
Card cosine	–	–	–	18.4 ± 0.0	19.2 ± 0.0	18.6 ± 0.0
z-score max (router, card cosine)	–	–	–	36.1 ± 0.7	15.3 ± 0.1	30.9 ± 0.6
z-score sum (router, card cosine)	–	–	–	40.3 ± 0.5	11.5 ± 0.4	33.1 ± 0.5
Rank max (router, card cosine)	–	–	–	32.2 ± 0.3	0.5 ± 0.1	24.3 ± 0.2
Rank sum (router, card cosine)	–	–	–	31.8 ± 0.3	5.1 ± 0.5	25.1 ± 0.4

B Card document

The frozen card embedder ϕ reads a plain-text document $d(m)$ built once at registration. The reported document (whole card) concatenates the catalog fields with the full stored description: domain, task (Hub pipeline tag), family, input/output signature, publisher, language, GPU tier, packaging, license, library, and the uncut card description. ϕ is not trained on CMRBench-Pref. Domain labels used for D-Acc remain the CMRBench domain field.

A short document (domain line plus the first 240 characters of the description) and a rich document (the same catalog fields with the description cut at 768 characters) can replace the whole card with no other change. Table 6 compares the three documents at 0.6B, 4B, and 8B. The margin rule in Figure 3 uses the 8B embedder on the whole card. Across the 464 models first seen in $\mathcal{E}_4$, description length has median 481 characters and 90th percentile 1501; the longest stored description is 18,648 characters. Whole-card token length at the 90th percentile is 450; none of the 464 documents exceeded the 8B context used for encoding.

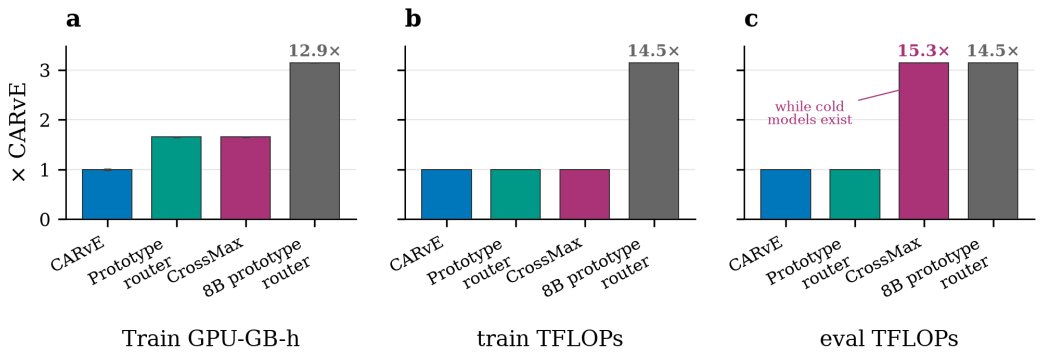


Figure 4: Cost relative to CARvE: (a) training GPU-GB-h, (b) training TFLOPs, (c) evaluation TFLOPs with the E4 models cold. Clipped bars are labelled with their true ratio.

Table 5: Replacing stored states with replayed exemplars (CMRBench-Pref, seed 40, after $\mathcal{E}_4$, prototype router without the preference penalty). *Full store* keeps every labelled query's state. *Exemplar* keeps the same training replay but rebuilds the class mean of every model that has a replay text: a model with queries in the incoming experience keeps those plus one live encoding of its replay text - a model without is scored from its replay text alone, $\mu(m) = \text{normalize}(Wh(\text{exemplar}))$, with no ancestor mix ($\kappa = 0$). Models without a replay text keep their stored states, so a smaller buffer leaves more rows unchanged. E2 texts is the replay buffer entering $\mathcal{E}_2$; one-exemplar rows counts models scored from a single replay text at that point.

Replay	E2 texts	One-exemplar rows	Full store	Exemplar	Gap
10%	782	672	47.2	46.5	−0.7
5%	391	347	47.7	47.3	−0.4
1%	78	69	43.1	43.8	+0.6

A larger card embedder helps once the card is rich. The card embedder can be swapped between 0.6B, 4B, and 8B with no retraining. Table 6 reports cold accuracy when the cold winner is always returned, on the same 920 cold queries. On a short card (the domain line plus 240 characters of description), going from 0.6B to 8B adds 2.2 points. On a rich card (the catalog fields plus 768 characters), it adds 6.5. The 0.6B embedder gains nothing from the richer card (28.2 against 28.3), so the extra text helps only a larger embedder. The margin rule uses the 8B embedder on the whole card (35.8); 0.6B and 4B were not run on that document.

C Protocol details

Training uses an effective batch of 128 (16 with 8 accumulation steps). Original-CMRBench cells use seeds 40, 41 and 42. Trained CMRBench-Pref rows (prototype router) and the CrossMax cells of Figure 3 use seeds 40, 41 and 42 (mean ± standard error). Retrieval baselines use seed 40 only as they are deterministic. Seeds propagate to weight initialisation, data order, herding selection, candidate construction and hard-negative mining.

Card embedder. Card vectors are written once at registration by a frozen Qwen3-Embedding encode of $d(m)$ (Appendix B). The reported card embedder is the 8B checkpoint; 0.6B and 4B use the same document with no gradient. The 8B and 4B encodes use the public SentenceTransformer query prompt; the 0.6B encode uses the project's last-token pooler.

Adaptation. Unless a table says otherwise, every trained cell uses LoRA of rank 32, α=64, dropout 0.05, on attention and MLP projections; maximum length 256; last-token pooling; effective batch 128. The Qwen3-Embedding-0.6B encoder is kept in fp32. The 8B prototype router uses the same LoRA, with the encoder in bf16, the router and loss in fp32, and adapter learning rate 10^{-5}.

Table 6: Cold accuracy when the cold winner is always returned, by card text and card-embedder size (920 cold queries).

Card text	0.6B	4B	8B
Short (240 chars)	28.2	30.1	30.4
Rich (768 chars)	28.3	32.9	34.8
Whole card	–	–	35.8

Continual regularisers. Replay, distillation, and anchoring are not a single recipe applied to every method. Table 7 records the headline settings. The prototype router's encoder uses the distillation term of Appendix D, weight 0.5, on herding replay, matching the pooled encoder state h to a snapshot of the previous encoder; this is not logit-level Learning without Forgetting (Li & Hoiem, 2018), which we do not use. Stored states receive no gradient, so there is no row anchor; the query projection is still pulled toward its previous value (λ=0.01). CARvE is a flat table with one learned vector per model, trained from that model's labelled queries. Its rows are neither class means nor card embeddings, so a model with no labelled queries has nothing to score. The replay rows of Table 3 compare this table with the detached class means at the same budget, though with different replay selection: herding for the prototype router and a domain–model coreset for CARvE. CARvE inherits the asymmetric anchoring of Bell et al. (2026): rows written by experience $t-1$ are held by a cosine penalty (λ=10^4), the projection by an MSE penalty (λ=5×10^4), and newly appended rows are free. CARvE does not use feature distillation. Replay into the CARvE encoder is a 10% domain–model coreset unless a table names another fraction.

C.1 Prototype lifecycle

Stored states. For each labelled training query of m, excluding replay, the checkpoint stores its pooled state $h(q)$, tagged with the experience that wrote it; $n(m)$ is the number of stored states. At every scoring pass the states are mapped by the current W and averaged after normalisation,

$$\bar{z}(m) = \mathrm{normalize}\Big(\frac{1}{n(m)}\sum_{q}\mathrm{normalize}\big(Wh(q)\big)\Big). \tag{8}$$

Because each projected state is normalised before averaging, a stored mean of the $h(q)$ would not determine $\bar{z}(m)$, so the individual states are kept.

Writing the states. In the experience $\mathcal{E}_t$ in which m is first labelled, its states are computed with the incoming encoder, the checkpoint from $\mathcal{E}_{t-1}$, so that m has a prototype throughout that experience. At the end of $\mathcal{E}_t$, the states written during $\mathcal{E}_t$ are dropped and each query is re-encoded once with the trained encoder. Later encoders never re-encode them; queries labelled in a later experience append their own states.

Training-time overwrite. Within a batch, the stored states of each gold model, including the gold model of a replayed query, are replaced by $n(m)$ copies of $\bar{h}$, an exponential moving average of its pooled states with momentum 0.2; while the copies agree, that model is scored as $\mathrm{normalize}(W\bar{h})$. For models first labelled in $\mathcal{E}_t$, the end-of-experience write restores one state per query. Models labelled in an earlier experience that appear as the gold model of a replayed query are not re-encoded and keep $n(m)$ copies of the average, so the per-query states of Eq. 8 no longer hold for them. All reported results use this behaviour; keeping the average in a separate buffer that never overwrites stored states is a direct alternative we have not evaluated.

Table 7: Continual regularisers in the headline results.

	Replay	Feat. distil.	Row anchor	Proj. anchor
Prototype router	10% herding	0.5 cosine	none (no gradient)	λ=0.01
CARvE	10% coreset	none	old, λ=10^4	λ=5×10^4

C.2 TAXONOMY ANCESTOR

Each model is placed in a four-level taxonomy derived from its CMRBench domain string and identifier: L0, the modality pair (text→text, image→label, ...); L1, the task category, i.e. the domain prefix (Natural Language Processing, Computer Vision, Audio, Multimodal, Tabular, Reinforcement Learning); L2, the task, i.e. the remainder of the domain string (text generation, object detection, ...); and L3, the structural family, i.e. the name cluster of the model identifier.

Node centroids are means of model query means in the space of z, filled from L3 upward. An L3 centroid is the support-weighted mean of its members; an L2 centroid is the unweighted mean of its child L3 centroids, an L1 centroid the mean of its L2 children, and an L0 centroid the mean of its L1 children. The ancestor $A(m)$ is the weighted mean of the four centroids on m's path, with weights $1/2$, $1/2$, 1, 1 on L0, L1, L2, L3. Because m belongs to its own nodes, $A(m)$ includes m itself.

With $\bar{z}(m)$ as in Eq. 8, the scored prototype shrinks it toward the ancestor,

$$\mu(m) = \mathrm{normalize}\big(\alpha(m)\,\bar{z}(m) + (1-\alpha(m))\,A(m)\big), \qquad \alpha(m) = \frac{n(m)}{n(m)+\kappa}, \quad \kappa = 2, \tag{9}$$

so a model with few queries leans on its ancestor and a model with many relies on its own mean.

D TRAINING LOSSES

Only the prototype router is trained. It minimises a cross-entropy of s_{pref} (Eq. 4) over the sampled candidate set of size $K{=}512$, plus the regularisers below. The weights are the headline settings of Table 7. The objective is

$$\mathcal{L} \;=\; \mathcal{L}_{\mathrm{CE}}(s_{\mathrm{pref}}) \;+\; 0.5\,\mathcal{L}_{\mathrm{distil}} \;+\; 0.01\,\mathcal{L}_{\mathrm{proj}}(W). \tag{10}$$

The class means receive no gradient. $\mathcal{L}_{\mathrm{distil}}$ is computed only on replayed queries. It matches the pooled encoder state h, the input of W in Eq. 1, to a snapshot h^{prev} taken at the start of the experience, before that map is applied. Writing $\mathcal{R}$ for the replayed queries in the batch and d for the width of h,

$$\mathcal{L}_{\mathrm{distil}} = \frac{1}{|R|}\sum_{q\in R}\Big[\big(1-\cos(h(q),h^{\mathrm{prev}}(q))\big) + \frac{1}{d}\sum_{i=1}^{d}\big(h_i(q)-h_i^{\mathrm{prev}}(q)\big)^2\Big] \tag{11}$$

The cosine is of the ℓ_2-normalised states. The second term is the mean of squared differences of the unnormalised coordinates. This is not a match of the projected vector $z(q)$. $\mathcal{L}_{\mathrm{proj}}(W)$ is the mean squared difference between the entries of W and the map saved at the end of the previous experience.

CARvE uses a cross-entropy of the scaled cosine of Eq. 2 on its flat per-model table, plus the two anchors of Table 7 and no distillation term. Newly appended rows are free.

E METRIC DEFINITIONS

Let $T{=}4$ and let $A_{t,j}$ be the accuracy after experience t evaluated on experience j. Headline accuracy in Table 3 is the mean of every cell on or below the diagonal (Rodríguez et al., 2018),

$$\mathrm{Acc} \;=\; \frac{2}{T(T+1)}\sum_{t=1}^{T}\sum_{j=1}^{t} A_{t,j}. \tag{12}$$

Model accuracy asks whether the returned model is the gold model. Domain accuracy asks whether the returned model and the gold model share a domain. Sat@1 asks whether the top-ranked model lies in the gold feasible set $F(c)$. Each experience pair (t, j) with $t{\geq}j$ contributes one cell, so the headline is neither the accuracy on the pooled queries nor the mean of the last row (Lopez-Paz & Ranzato, 2017). Retrieval rows have a single checkpoint, so every row of their triangle repeats that checkpoint's accuracy on each experience. Joint rows see all four experiences at once and report the mean over the four evaluation sets of that checkpoint.

Forgetting uses the same matrix. The reported drop is from the diagonal to the final row, averaged over the earlier experiences,

$$\mathrm{Fgt} \;=\; \frac{1}{T-1}\sum_{j=1}^{T-1}\big(A_{j,j} - A_{T,j}\big). \tag{13}$$

$A_{j,j}$ is the accuracy just after experience j was learned, and $A_{T,j}$ is the accuracy on that same experience after $\mathcal{E}_4$. Model and domain forgetting substitute the corresponding accuracy. A positive value is a drop; Lopez-Paz & Ranzato (2017) report the same quantity with the opposite sign, as backward transfer. Means and standard errors are over seeds 40, 41 and 42.

Warm, cold, and mixed accuracy do not use Eq. 12. They are model accuracy on query sets of the after-$\mathcal{E}_3$ pool grown to $\mathcal{E}_4$: queries whose gold model is warm, queries whose gold model is cold, and their union. In Table 2, Sat@k is the fraction of the top-k that lies in the feasible set of that row. Rows above the swap use the gold preference c, and the Swap GPU row uses the swapped preference. All columns of Table 2 use the triangle mean of Eq. 12, as in Table 3.

F CMRBench-Pref construction

CMRBench-Pref is obtained from CMRBench by writing an English preference prefix in front of the original request. The prefix is always feasible for the gold model. Task, publisher, and language are fields the card already has. Size is not: the card stores a parameter count, and the prefix names a GPU. All 34,239 rows are constrained (median $|F(c)| = 44$). Fields sampled into the prefix are task (30,978), GPU (24,884), publisher (5,431), language (2,162), and packaging (131). Language is written only when the request is in or about that language. A model satisfies the GPU field when its estimated inference footprint fits the named tier. The sampler keeps the tightest tier the gold model already fits (12GB, 24GB, 48GB, 80GB, or multi-GPU). The wording does not always name the tier: 3,949 GPU prefixes name neither a memory size nor a card. Of these, 1,872 say "a consumer GPU", all from the 12GB tier, and 2,077 say "one GPU", of which 210 come from the 24GB tier and the rest from the 12GB tier. The parameter count is never written into the prefix. Packaging that means a local GGUF file is kept with probability 0.15 and phrased as local or at-home, never as GGUF. Missing Hub tags are inferred from the CMRBench domain.

The sampler prefers two-field prefixes; one- and three-field prefixes are allowed. A GPU-only or packaging-only prefix is rejected when another field exists. Each prefix is first stitched from a template. A language-model rewrite is kept only when an automatic audit accepts it: the rewrite must keep the original goal, include every listed preference, invent no GPU, publisher, or language, and avoid Hub jargon. Of the 34,239 rows, 17,375 are first-pass rewrites, 5,573 are retries, and 11,297 keep the template. The audit passes on 0.999 of rows (36 identity failures; no coverage failures). Because the task is sampled into about 31k queries, domain accuracy on CMRBench-Pref is close to reading a task label off the input, and model accuracy is the quantity of interest.

G Preference penalty and extracted $\hat{c}$

The preference penalty of Eq. 4 uses five fields: task, GPU, publisher, language, and packaging. Each field owns a value table C_f of dimension 16. Each card is indexed by the tightest GPU tier its estimated footprint fits, under the same estimate as Appendix F (12GB if that estimate allows it, otherwise 24GB, 48GB, 80GB, or multi-GPU). Publisher is a speakable brand name (Google, Meta, ...). A missing field contributes the zero vector. The scale a is a learned scalar, initialised at 1. The card embedder and CARvE have no C_f tables.

The tables are trained on the gold preference c that generated the prefix. Evaluation does not read c. Instead, an extracted preference $\hat{c}$ is recovered from the query text: a lexicon for GPU, publisher, language, and local packaging, plus a two-layer MLP task head fitted on pooled encoder states of the training split visible at that checkpoint (60 epochs, learning rate 10^{-2}). The head's prediction is accepted at confidence 0.7; if a longest-phrase match disagrees, the task is omitted. Vague GPU language ("one GPU", "a single GPU") is mapped to the 12GB tier, matching how cards are indexed in the value tables; this mapping is wrong for the 210 "one GPU" prefixes generated from the 24GB

tier (Appendix F). Sat@1 is always whether the resulting top-ranked model lies in the gold $F(c)$, and Sat@k is the fraction of the top-k that lies in $F(c)$.

After $\mathcal{E}_4$ the cold group is empty, and the prototype router (and therefore CrossMax) scores with Eq. 4 using this $\hat{c}$ and no hard mask. That is the readout of Table 3. On the curve of Figure 3, the warm winner is the plain class-mean cosine, without the preference penalty. The cold winner is the best cold model under the frozen cosine of Eq. 5, inside the extracted feasible set $F(\hat{c})$; an empty set falls back to the unmasked ranking. Table 2 probes the preference penalty. Dropping the prefix removes the penalty. The GPU swap keeps only the GPU field, rewrites it to a tier the gold model fails, and keeps the query only when another model meets that tier.

Parse-then-filter. Table 3 reports two parse-then-filter rows, both built on the prototype router trained on the original CMRBench requests and scored on CMRBench-Pref queries. In each, a regex parse of the preference prefix gives $\hat{c}$, using longest-phrase matching for GPU, publisher, language, and packaging, plus Hub task phrases, and the ranking is then masked to $F(\hat{c})$. An empty $F(\hat{c})$ falls back to the unmasked ranking. This is a hard mask after scoring, not the preference penalty of Eq. 4. The two rows differ only in what the router encodes. In the first, the prefix stays in the query, which is text this router never saw during training. The mask moves M-Acc from 34.3 to 31.7 and Sat@1 from 76.0 to 80.8, and an oracle $F(c)$ mask gives 39.9. The second is the comparison used in the main text. The prefix is parsed into $\hat{c}$ and then stripped from the query before ranking (seed 40). The mask moves M-Acc from 42.2 to 41.9 and Sat@1 from 75.3 to 84.8, and an oracle $F(c)$ mask gives 52.2. In both settings the parsed filter raises Sat@1 but not accuracy, and it stays below the 46.9 of the router trained on preference-bearing queries. The oracle rows show that most of the remaining gap comes from errors in $\hat{c}$ rather than from filtering itself.

H THE MARGIN AND THE SECOND ENCODE

The conservative δ is calibrated on an earlier stage of the stream, never on the evaluated one. We take the $\mathcal{E}_2$ checkpoint, register the $\mathcal{E}_3$ models as cold, and compute the card margin of Eq. 7 on the $\mathcal{E}_3$ queries. δ is the 80th percentile of that distribution, i.e. the margin that would have returned the cold winner on 20% of $\mathcal{E}_3$ queries. That δ is then fixed and applied unchanged to the after-$\mathcal{E}_3$ pool grown to $\mathcal{E}_4$, where it returns the cold winner on 18.8% of queries (Table 4).

After $\mathcal{E}_4$ every model is warm, so $\mathcal{M}_t^{\text{cold}}$ is empty and CrossMax costs one 0.6B encode per query, the same as the prototype router. While some models are cold, each query is encoded by the prototype router's 0.6B encoder and by the frozen 8B card embedder, and each card is encoded once at registration. On the $\mathcal{E}_1$–$\mathcal{E}_3$ evaluation queries plus the $\mathcal{E}_4$ cold queries, the evaluation cost in Figure 4 is one 0.6B query encode plus one 8B query encode. The 0.6B encode costs 250.9 GFLOP and the frozen Qwen3-Embedding-8B encode is $15.3\times$ one 0.6B encode.