POLLINATOR: OPTIMAL MATCHMAKING IN AN INTELLIGENCE MARKETPLACE

Anonymous authors
Paper under double-blind review

ABSTRACT

The rapid growth of the intelligence marketplace has created an abundance of Large Language Model (LLM) producers, each with different cost–performance tradeoffs, making optimal selection challenging and resource-intensive. We present POLLINATOR, a novel router that integrates a frugal, data-efficient predictor with an online dual-based optimizer. The predictor combines graph-based semi-supervised learning with an Item Response Theory (IRT) head, reducing training cost by up to 49% while improving predictive accuracy over prior state-of-the-art. The optimizer formulates matchmaking as a strongly convex problem, which allows efficient dual-to-primal conversion for real-time serving. Extensive experiments demonstrate that POLLINATOR delivers superior cost–performance tradeoffs: achieving 0.43%-1.5% gains at 71%-93% of the cost of state-of-the-art router, 3-5% gains at only 1.9-3% of the cost of the best individual producer, and up to 10.6% higher accuracy at just 0.3-35.7% of the cost on challenging real-world benchmarks such as BFCL-V3 and MMLU-Pro. Finally, the interpretability of learned query difficulties and model abilities demonstrates POLLINATOR's effectiveness for dynamic and cost-efficient intelligence matchmaking.

1 INTRODUCTION

The Intelligence Marketplace. The commoditization of (artificial) intelligence, which gave birth to the phrase *Intelligence on Tap*, coupled with the rapid proliferation in applications that exploit it as a *Design Material* Holmquist (2017), catalyzed a Cambrian explosion of intelligent applications. The author of such an application, however, faces a problem of plenty: there are many producers of intelligence[1] with varying cost-performance tradeoffs on generic benchmarks, making it hard to choose an optimal one appropriate specific to the application at hand. Furthermore, given the frequent updates that alter performance and rapidly falling costs, the process of optimization has to be repeated continuously – thus putting a constant demand on the author's resources.

Matchmaking with Router. To remedy, *router* – which routes each request independently to a producer from a pool based on projected cost-performance tradeoff – was conceptualized Hu et al. (2024). A canonical router consists of two components: a collection of *predictors* that project the cost and performance for each request-producer pair; and, an *optimizer* that makes the tradeoff based on the projection.

Predictor & Data Efficiency. While a wide gamut of predictors, ranging from simple k-Nearest Neighbor Hu et al. (2024) to sophisticated Small Language Model with bespoke Bradley-Terry head Ong et al. (2024) and Item Response Theory Song et al. (2025), have been explored, the angle of data-efficiency has remained underexplored. Recently, Tsiourvas et al. (2025) approached data-efficiency in predictor through the lens of causal inference. However, we remark that the causal-inference setting is overly restrictive: one can indeed send the same request (unit) to more than one producers (treatments) in order to gauge cost and performance (treatment effects), which is a departure from the assumptions in causal-inference. In this work, we combine the superior predictive

[1] A search in *Hugging Face* for Transformer-based models with 3B+ parameters results in 170K+ hits. The benchmarking service, *Artificial Analysis*, indexes 250 frontier models.

performance rendered by Item Response Theory grounded in psychometry, with the data-efficiency afforded by graph-based semi-supervised learning to design POLLINATOR, a novel data-efficient predictor.

Optimizer & Online Matchmaking. The prevalent approach to matchmaking in the literature has been to compute the *utility* of each producer for the given request by blending the predicted performance and cost either with a affine combination (with *willingness-to-pay* as a hyper-parameter) Hu et al. (2024), or with a convex combination (again a hyper-parameter) Song et al. (2025). Along similar lines, Somerstep et al. (2025) advances the frontier by designing rate-optimal predictors for cost and performance. Mei et al. (2025) frames the problem as a Linear Program, and recovers the primal solution from the dual variables. Taking inspiration from an optimizer designed for two-sided marketplace Agarwal et al. (2012), we choose to frame the optimization problem as a strongly convex program, which simplifies the dual-to-primal conversion to facilitate online serving. See Appendix B for additional related works.

Contributions. In summary, we make the following contributions: ❶ POLLINATOR implements its predictors atop Graph Convolutional Network Kipf (2016) with a novel Item Response Theory (IRT)-based head that cuts down the training cost by 49%, while surpassing the predictive performance of the vanilla IRT-based predictor proposed in Song et al. (2025); ❷ the (Lagrangian) dual-based online serving scheme, coupled with the strongly convex primal program delivers superior cost-performance tradeoff than the Linear Program-based serving scheme Song et al. (2025) by boosting performance by 0.43%-1.5% at 71%-93% of the cost; ❸ in the in-domain and out-of-domain setting proposed in Song et al. (2025), POLLINATOR yields 3-5$\%$ boost in performance at mere 1.9-3$\%$ of the cost of the best producer; ❹ report superior performance in two novel settings on real-world and contemporary benchmarks, such as BFCL-V3 (tool-calling) and MMLU-Pro, where POLLINATOR achieves 3-10.6$\%$ higher accuracy at only 0.3-35.7$\%$ of the best producer's cost.

2 INTELLIGENCE MARKETPLACE & THE MATCHMAKING PROBLEM

Background. The *matchmaker* dispatches each inference request emanating from the consumer, enumerated with $i \in [M]$, in a just-in-time fashion to a request-specific optimal producer, $j \in [N]$. Upon receiving the response, we assume that the matchmaker possesses the ability to compute a *ex post* quality – such as accuracy, $a_{ij} \in \mathbb{R}_+$ – as well as the resource consumption – such as cost, $c_{ij} \in \mathbb{R}_+$ and latency $t_{ij} \in \mathbb{R}_+$ – metrics. Furthermore, we assume that the matchmaker possesses the corresponding *ex ante* estimates, $\hat{a}_{ij}$, $\hat{c}_{ij}$ and $\hat{t}_{ij}$, *before dispatching the request*. Equipped with the ex ante estimates, informally, the role of the matchmaker is to maximize the sum total of response quality, while obeying guardrails on total inference cost, and possibly other resource consumption metrics. We now frame the matchmaker's optimization problem formally.

Optimal Matchmaking. At this point, we distinguish between two settings. First, *batch inference*, where all the inference requests, $i \in [M]$, are known *a priori*. This formalizes the setting a certain class of consumers operate in, e.g., document summarization and information extraction. We posit the matchmaker's objective as to maximize the total ex ante response quality, $\sum_{i\in[M]} \sum_{j\in[N]} x_{ij}\hat{a}_{ij}$, while obeying a guardrail on total inference cost, $\sum_{i\in[M]} \sum_{j\in[N]} x_{ij}\hat{c}_{ij} \leq C$, where x_{ij} is the collection of *primal* variables lying on the probability simplex, Δ^N, defined with the constraints $x_{ij} \geq 0, \forall i \in [M], \forall j \in [N]$ and $\sum_{j\in[N]} x_{ij} = 1, \forall i \in [M]$. We remark that the consumer may want to impose additional constraints, such as minimum volume commitments, where each producer is guaranteed to receive a specified minimum volume of inference request, $\sum_{i\in[M]} x_{ij} \geq M_j, \forall j \in [N]$. The second setting is *online inference*, where the inference requests arrive in a stream, along with the corresponding ex ante estimates. Specifically, at time i, the decision $\mathbf{x}_i \in \Delta^N$ has to be taken, *without a foreknowledge of the upcoming requests*, $\mathbf{x}_k, \forall k > i$. The ex post cost and quality are defined, in this case, over a long horizon, M. [2] Lastly, the matchmaker will be required to follow a reference policy, q_{ij}, where the desired level of proximity is expressed as $\frac{1}{2}\gamma \sum_i \sum_j (x_{ij} - q_{ij})^2$. In order to ease the exposition, we now focus on the setting where quality

[2] We remark that in practice, certain additional guardrails become desirable in the online setting: e.g., on $p95$ latency – which can also be specified as a linear constraint, in terms of Conditional Value-at-Risk (CVaR).

and the cost are the only constraints at play. We remark that our framework extends to a more general setting, and can incorporate additional linear constraints, such as minimum volume commitments and $p95$ latency. Thus, the canonical form the matchmaker's optimization problem assumes can be expressed as follows.

Definition 1 (Optimal Matchmaking)**.**

$$\min_{\mathbf{x}\in\Delta^N} \frac{1}{2}\gamma\|\mathbf{x}-\mathbf{q}\|_F - \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij}\hat{a}_{ij} \text{ s.t. } \sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij}\hat{c}_{ij} \leq C \tag{1}$$

Note that Eq. 1 succinctly describes the online version of the problem as well, assuming C is the long-horizon budget applicable over a span of M requests. Before turning our attention to the solution of Eq. 1, we lay down the design desiderata for the POLLINATOR – the novel optimal matchmaker presented in the present work.

Desiderata. In order to ensure practicality of the POLLINATOR system, we impose 2 design desiderata: ❶ *Frugality.* The cost savings yielded by the matchmaker during inference must not be offset by the cost of training its predictors; ❷ *Safety.* For a consumer to be able to relinquish its control over the choice of the producer, it must be assured adherence to the specified guardrails. We now detail how these desiderata guide the design of POLLINATOR.

2.1 GRAPH-BASED SEMI-SUPERVISED LEARNING & ENSURING FRUGALITY

We design a two-tower architecture reminiscent of recommendation system: the first tower encodes the request emanating from the consumer; the second tower encodes the producers; while a combiner combines the output of the two towers and emits the ex ante estimates. We detail each of the components below.

Request & Producer Towers. The prompt in request i, $p_i \in \Sigma^*$, is first encoded into a dense vector, $x_i := \texttt{Enc}^{\texttt{R}}(p_i)$, where $\texttt{Enc}^{\texttt{R}} : \Sigma^* \mapsto \mathbb{R}^d$ is a pre-trained BERT encoder on vocabulary Σ, that extracts the embedding corresponding to the [CLS] token that is appended to the prompt. On the set of prompts in the training dataset, we then induce a k-nearest neighbour graph, $\mathcal{G}(\mathcal{V},\mathcal{E})$, where the similarity between nodes $v_i, v_j \in \mathcal{V}$ is defined in terms of the cosine similarity of their corresponding embeddings, $\cos(\measuredangle x_i x_j)$. Let A denote the adjacency matrix of this graph, and let $\tilde{A} := A + I$ denote the adjacency matrix of $\mathcal{G}$, with self-loops added. On this graph, the request tower implements a Graph Convolutional Network (GCN), which propagates information between two successive layers, l and $l+1$, with $H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$. The embedding of node v_i at layer $l = 0$ is set to its embedding, $H_i^{(0)} = x_i$. Lastly, the embedding of the node v_i from the last layer of the GCN, $H_i^{(L)}$, is mapped to a vector, $\alpha_i \in \mathbb{R}^D$, and a scalar, $\beta_i \in \mathbb{R}$, via two learnable linear projections, W_α, W_β, respectively. Similarly, each producer, $j \in [N]$, is mapped to a embedding, $\theta_j \in \mathbb{R}^D$, with an encoder, $\texttt{Enc}^{\texttt{P}} : [N] \mapsto \mathbb{R}^D$. In our experiments, the encoder is a pre-trained BERT that encodes the textual description of the producer, or a simple lookup-based learnable linear projection, W_θ.

Combiner. Inspired by Item Response Theory (IRT), the combiner treats α_i as the *discrimination* parameter, which intuitively models the skill-set required to generate a high-quality response to request, i, and treats β_i as its *difficulty*. On the other hand, θ_j models the skill-set offered by the producer j. IRT posits that the probability of obtaining a high-quality response improves with the degree of match between the skill-set required to process request i, and those offered by producer j – and is modulated by the difficulty of the request. This intuition is operationalized as: $\mathbb{P}\{Y_{ij} = 1 := \sigma(\alpha_i^T\theta_j - \beta_i)\}$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the usual sigmoid function.

Training & Inference. During training, the performance predictor is fit by minimizing the binary cross-entropy loss. The cost estimate is simply taken to be the average cost in training dataset. During inference, we first induce a graph among the incoming request and its k nearest neighbors in the training dataset, and then run the forward-pass for both the towers. Figure 3 in Appendix A.1 summarizes the salient workflows.

Algorithm 1 Dual Serving Scheme

Input: Request i; Dual λ.
Output: Primal serving scheme $\{x_{ij}\}_{j=1}^{N}$.
Fetch ex ante predictions $\{\hat{a}_{ij}\}_{j=1}^{N}$ and $\{\hat{c}_{ij}\}_{j=1}^{N}$ ▷ Invoke predictors.
Compute utilities, $\{u_{ij}\}_{j=1}^{N}$, and sort them into $u_{i(1)}, \cdots, u_{i(N)}$ ▷ Compute and sort utilities.
$U = \gamma$, $j = 1$
repeat
  if $u_{i(j)} + \frac{U - u_{i(j)}}{j} \leq 0$ **then**
    $j = j - 1$; **break**
  else
    $U = U - u_{i(j)}$; $j = j + 1$; **continue**
  end if
until $j \geq N$
$\nu_i = \frac{U}{j}$
$x_{u(k)} = \frac{u_{i(k)} + \nu_i}{\gamma}, \forall k \leq j; x_{u(k)} = 0, \forall k > j$

2.2 Dual-based Optimization & Adherence to Safety

The primal optimization problem formulated in Eq. 1 comprises a strongly-convex objective and several linear constraints, thus rendering several off-the-shelf solvers immediately applicable at a first glance. However, in practice, its deployment faces two challenges.

Predict-and-Optimize. The first problem arises because of the predict-and-optimize paradigm. Ideally, the primal problem in Eq. 1 should have been defined in terms of the ex post coefficients, a_{ij} and c_{ij}. However, we only have access to the corresponding ex ante coefficients, $\hat{a}_{ij}$ and $\hat{c}_{ij}$. Thus issues such as predictor's inaccuracy and mis-calibration plague the constrained optimization via both constraints and the objective. We address this issue via improving the accuracy and the calibration of the predictors, and leave a more principled investigation based on the predict-then-optimize framework for a future work.

Online Optimization. The second problem appears in the case of online inference. Without *a priori* knowledge of the M requests, the primal problem cannot even be formulated. POLLINATOR solves it via resorting to the *dual*. The Lagrangian of Eq. 1 is presented in Eq. 2.

$$\min_{\mathbf{x} \in \Delta^N} \max_{\lambda, \delta \geq 0, \nu} \frac{1}{2}\gamma \|\mathbf{x} - \mathbf{q}\|_F - \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{N} x_{ij} \hat{a}_{ij} + \sum_{i=1}^{M} \sum_{j=1}^{N} \lambda (x_{ij} \hat{c}_{ij} - C) - \sum_{i} \nu_i \left(\sum_{j} x_{ij} - 1\right) - \sum_{i} \sum_{j} \delta_{ij} x_{ij} \tag{2}$$

The *stationarity* condition amongst the Karush-Kuhn-Tucker (KKT) conditions allow us to express the primal solution in terms of the dual variables as, $x_{ij} = \frac{u_{ij} + \nu_i + \delta_{ij}}{\gamma}$, where the *utility*, $u_{ij} := \gamma q_{ij} + \frac{1}{M}\hat{a}_{ij} - \lambda \hat{c}_{ij}$. However, this still does not yield a serving plan, given the presence of request-dependent dual variables, ν_i and δ_{ij}, in the numerator.

In order to get rid of the request-dependent dual variables, we need to appeal to the *complementary slackness* condition amongst the KKT conditions, which leads to the following proposition.

Proposition 1 (Utility)**.** *In the optimal solution for request i, assume producer j_1 has more utility than producer j_2, $u_{ij_1} \geq u_{ij_2}$. If $x_{ij_2} > 0$, then $x_{ij_1} \geq x_{ij_2} > 0$.*

When the request i arrives, armed with the ex ante estimates, $\hat{a}_{ij}$ and $\hat{c}_{ij}$, and the request-independent dual variable, λ, we first rank the producers as per their respective utilities. Let $u_{i(1)}, \cdots, u_{i(N)}$ represent the ranked list, where $u_{i(1)} \geq \cdots \geq u_{i(j^*)} \geq \cdots \geq u_{i(N)}$. Proposition 1 illuminates a structure in the solution: $x_{i(1)} \geq \cdots \geq x_{i(j^*)} \geq \cdots = x_{i(N)}$, where $x_{i(j^*)}$ is the smallest non-zero element. Thus, complementary slackness ensures $\delta_{i(j)} = 0$, when $j \leq j^*$, and, primal feasibility ensures, $\sum_{j=1}^{j^*} x_{i(j)} = 1$. This insight allows us to eliminate the last remaining request-specific dual variable, ν_i, from the expression for the primal solution x_{ij}, by ensuring

$\nu_i = \frac{\gamma - \sum_{j=1}^{j^*} u_{i(j)}}{j^*}$. Proposition 2 distils this insight into a operational definition of j^*, which culminates into the optimal primal serving scheme presented in Algorithm 1.

Proposition 2 (Optimal Stopping). *j^* is the maximum j such that $u_{i(j)} + \frac{\gamma - \sum_{k \le j} u_{i(k)}}{j} > 0$.*

3 Experimental Setup

3.1 Benchmark & Implementation

In-Domain Dataset (ID). Following Song et al. (2025), we evaluate PolliNator's in-domain generalization capability over 8 datasets: ① **MMLU** (Hendrycks et al., 2020) (reasoning and knowledge across 57 domains), ② **CMMLU** (Li et al., 2024) (a Chinese incarnation of MMLU), ③ **ACLUE** (Zhang & Li, 2023) (ancient Chinese language-understanding), ④ **ARC_C** (Clark et al., 2018) (advanced reasoning), ⑤ **HotpotQA** (Yang et al., 2018) (question-answer requiring multi-hop reasoning), ⑥ **SQuAD** (Rajpurkar et al., 2018) (reading comprehension), ⑦ **MATH** (Hendrycks et al., 2021) (competition-level mathematics), and, ⑧ **MBPP** (Austin et al., 2021) (basic coding). 20 LLMs were in the candidate pool (see Table 7 in Appendix A.3). PolliNator is trained on 70% of the *combined* dataset, and tested in-domain on the remaining 30%.

Out-of-Domain Dataset (OOD). Following Song et al. (2025), we evaluate PolliNator's out-of-domain generalization capability over 4 datasets that we not part of the training: ⑨ **CEVAL** (Huang et al., 2024) (tasks in Chinese language), ⑩ **CommonsenseQA** (Talmor et al., 2019) (commonsense reasoning), ⑪ **GSM8K** (Cobbe et al., 2021) (grade-school mathematics), and, ⑫ **HumanEval** (Chen et al., 2021) (coding). The same candidate LLM pool as In-Domain Dataset (**ID**) is employed. We emphasize that 100% of the combined datasets are used in test.

MMLU-Pro & BFCL-V3. In order to further evaluate PolliNator's efficacy on contemporary datasets and real-world scenarios such as tool-call, we benchmark in-domain on 2 additional datasets: ⑬ **MMLU-Pro** (Wang et al., 2024) (extends **MMLU** with harder multiple-choice questions with 10 possible choices, instead of 4), and, ⑭ **BFCL-V3 (Simple)** (Patil et al., 2025) (reasoning and ability to call external tools and APIs in real-world setting – a key skill for agentic applications). The LLM candidate pool consists of 15 members, including GPT and Gemini families, for **MMLU-Pro** (details in Table 9 in Appendix A.4), and 10 for **BFCL-V3 (Simple)**, including OpenAI o-series and Llama-3.1 families (see Table 10 in Appendix A.4 for an exhaustive list). For each of these 2 datasets, 70% is used for training and the rest 30% for testing (we emphasize that we do not combine the datasets). Table 6 in Appendix A.2 enumerates comprehensive details on all 14 datasets, including evaluation metrics and dataset cardinality.

Implementation. We use `bert-base-uncased` [3] as the request encoder, $\texttt{Enc}^{\texttt{R}}$. The producer encoder, $\texttt{Enc}^{\texttt{P}}$, simply encodes the producer ID, $j \in [M]$. The GCN consists of $L = 2$ layers with hidden dimension of 64 and dropout rate of 0.3. The producer's skill-set, θ_j, has dimension of 16. The kNN graph construction uses $k = 3$ nearest neighbors, with edge weights (optionally) set to the cosine similarity, $\cos(\measuredangle x_i x_j)$. PolliNator is trained with the Adam optimizer (Adam et al., 2014), with learning rate 1×10^{-3} and weight decay 1×10^{-5} for 200 epochs. During inference, we induce a graph among the test request and its $k = 3$ nearest neighbors. All hyper-parameters were selected based on best performance on a held-out validation set. All experiments are run on 1 NVIDIA A40 GPU with 40GB memory.

3.2 Metrics & Baselines

Metrics. We evaluate routing algorithms by the cost-performance tradeoffs they offer in the test dataset. Given binary ground-truth label, $a_{ij_i} \in \{0, 1\}$, where j_i is the chosen producer for request i, **Performance (%)** is defined in a dataset-dependent manner as either of *accuracy*, *F1*, *Exact Match (EM)* and *Pass@1* over the test dataset (see Table 6 in A.2 for dataset-specific evaluation metrics) – a methodology that echoes Song et al. (2025). **Cost ($)** is calculated as per the token counts and the rate-card for the producer j_i, where i indexes the requests in test dataset.

[3] `https://huggingface.co/google-bert/bert-base-uncased`

Baselines. We compare POLLINATOR against a set of strong baselines: ① **Small LLM** always routes queries to the smallest candidate model. ② **Large LLM** always routes queries to the largest candidate model. ③ **kNN-Router** is a simple retrieval-based baseline that selects an LLM based on the top-k ($k = 5$ gave the best performance) most similar queries, choosing the lowest-cost model among them. ④ **HybridLLM** (Ding et al., 2024) trains a pre-trained encoder (DeBERTa-v3) with matrix factorization to decide between a small and a large model. ⑤ **RouteLLM** (Ong et al., 2024) uses a binary classifier for pairwise LLM routing. ⑥ **RouterBench** (Hu et al., 2024) provides multiple routing strategies; we adopt its Predictive Router variant to avoid the high cost of querying all candidate models. ⑦ **MIRT-Router** and **NIRT-Router** (Song et al., 2025) are IRT-based methods and serve as our closest baselines. We implement only kNN-Router; results for the others are taken from Song et al. (2025). MIRT-Router is run on MMLU-Pro and BFCL-V3 using their official code, while NIRT-Router is omitted due to its reliance on costly GPT-4o relevance vectors. Following Song et al. (2025), we also generate LLM profile descriptions (Table 13 in Appendix A.8) for candidate models in **MMLU-Pro** and **BFCL-V3 (Simple)**.

4 RESULTS

In **ID** dataset, as seen in Table 1, in the performance-oriented batch inference setting, POLLINATOR delivers 0.8% performance gain over NIRT-Router at 70% of its cost. Similarly, in the cost-oriented setting, POLLINATOR renders 33% cost reduction over MIRT-Router, at a slightly better performance. In the balanced setting, POLLINATOR achieves a 0.9% gain over SOTA at 93% of the cost, and a 5% improvement over the best producer at 3% cost. In **OOD** (Table 2), the relative cost reduction in the balanced setting stands at 28.57%, at a similar performance. The $\geq 25\%$ cost advantage holds on MMLU-Pro and BFCL V3 datasets as well with 1.5% and 0.43% gains over SOTA, as seen from Table 3. Overall, POLLINATOR delivers a superior Pareto frontier in the cost-performance plane across ID, OOD, and real-world benchmarks . POLLINATOR effectively routes queries to the most appropriate model, avoiding unnecessary invocation of expensive LLMs while respecting global budget constraints.

Table 1: Comparison of routing methods in In-Domain Dataset across three distinct performance-cost tradeoff scenarios. **Bold** and underline denote the best and second-best results.

Method	Performance-First		Balanced		Cost-First	
	Performance (%)↑	**Cost ($)↓**	**Performance (%)↑**	**Cost ($)↓**	**Performance (%)↑**	**Cost ($)↓**
Small LLM	48.70	**0.31**	48.70	**0.31**	48.70	0.31
Large LLM	77.53	12.93	77.53	12.93	77.53	12.93
HybridLLM	54.37	1.98	52.42	1.54	56.65	2.78
RouteLLM	77.25	12.80	73.59	11.15	66.24	7.51
RouterBench	80.01	1.15	79.48	0.53	78.36	0.37
kNN-Router	74.38	1.14	74.38	1.14	74.38	1.14
MIRT-Router	80.67	0.42	80.65	0.42	80.03	0.39
NIRT-Router	80.69	0.55	80.41	0.43	79.37	0.41
POLLINATOR	**81.38**	0.39	**81.38**	0.39	**80.09**	**0.26**

Table 2: Comparison of routing methods in Out-of-Domain Dataset.

Method	Performance-First		Balanced		Cost-First	
	Performance (%)↑	**Cost ($)↓**	**Performance (%)↑**	**Cost ($)↓**	**Performance (%)↑**	**Cost ($)↓**
Small LLM	59.83	**0.11**	59.83	0.11	59.83	0.11
Large LLM	84.90	5.30	84.90	5.30	84.90	5.30
HybridLLM	63.34	0.73	62.08	0.41	63.79	0.65
RouteLLM	84.39	5.25	79.90	4.74	75.06	3.48
RouterBench	85.50	0.26	85.75	0.16	84.62	0.12
kNN-Router	80.92	0.29	80.92	0.29	80.92	0.29
MIRT-Router	87.12	0.14	87.12	0.14	87.18	0.13
NIRT-Router	**87.37**	0.15	87.24	0.14	87.46	0.13
POLLINATOR	**87.37**	0.14	**87.63**	**0.10**	**87.69**	**0.09**

Table 3: Comparison of routing methods on MMLU-Pro and BFCL V3 (ToolCall) datasets under the Performance-First setting, reporting model accuracy and associated cost.

Method	MMLU-Pro		BFCL-V3 (ToolCall)	
	Performance (%)↑	**Cost ($)↓**	**Performance (%)↑**	**Cost ($)↓**
Small LLM[1]	53.07	2.10	77.33	0.01
Large LLM[2]	71.61	5.01	89.33	1.85
kNN-Router	74.23	2.55	86.67	0.02
MIRT-Router	<u>78.84</u>	<u>1.18</u>	<u>90.66</u>	<u>0.008</u>
POLLINATOR	**79.18**	**0.88**	**92.00**	**0.006**

5 ABLATION STUDY

We perform a comprehensive ablation study on the performance predictor inside POLLINATOR on **ID** datasets to assess the impact of key design decisions, encompassing: ① request encoder, $\texttt{Enc}^{\texttt{R}}$; ② choice of k in graph construction; ③ the dimensionality, D, of $\theta_j \in \mathbb{R}^D$; ④ fraction of labeled nodes in GCN. Findings are summarized in Table 4 and Table 5. For detailed ablations across all combinations of $\texttt{Enc}^{\texttt{R}}$, k, D, and edge weighting strategies, refer to Table 12 in Appendix A.7.

Table 4: Ablation study of the predictor (without optimizer). Accuracy (%) is reported for different request encoders, graph neighborhood sizes, and varied model ability dimension. POLLINATOR is robust to embedding and neighborhood choices, sensitive to model ability dimension. The best configuration of POLLINATOR is marked with †. Neighborhood size and model ability experiments use *bert-base-uncased* as $\texttt{Enc}^{\texttt{R}}$.

Request Encoder ($\texttt{Enc}^{\texttt{R}}$)

$\texttt{Enc}^{\texttt{R}}$	Perf. (% ↓)
bert-base-uncased†	82.07 (-)
all-MiniLM-L6-v2	80.75 (↓1.32)
all-mpnet-base-v2	81.70 (↓0.37)
text-embedding-3-large	81.34 (↓0.73)

Graph Neighborhood Size (k)

k	Perf. (% ↓)
3†	82.07 (-)
5	81.92 (↓0.15)
10	81.87 (↓0.20)
20	82.02 (↓0.05)

Model Ability Dimension (D)

D	Perf. (% ↓)
3	71.43 (↓12.96)
5	80.49 (↓1.58)
8	79.26 (↓2.81)
16†	82.07 (-)
25	81.96 (↓0.11)
35	80.52 (↓1.55)
45	80.25 (↓1.82)

Request Encoder $\texttt{Enc}^{\texttt{R}}$. We experiment with different $\texttt{Enc}^{\texttt{R}}$ to encode requests. The best configuration (POLLINATOR†) achieves 82.07% accuracy. Alternatives such as *all-MiniLM-L6-v2*[4], *all-mpnet-base-v2*[5], and t*ext-embedding-3-large*[6] show a slight drop in performance of 1.32%, 0.37%, and 0.73% respectively, as reported in Table 4 (left), indicating that the predictor is robust to the choice of embedding while preserving strong predictive capability.

Neighborhood Size (k). We analyze predictor performance under varying graph neighborhood sizes k. The best accuracy occurs at $k = 3$ (82.07%), while larger neighborhoods ($k = 5, 10$) reduce accuracy (81.92%, 81.87%), and $k = 20$ only partially recovers it (82.02%). This indicates that large k introduces noisy neighbors, limiting predictor precision (Table 4, middle).

Effect of Model Ability Dimension (D). We assess how the model ability dimension D impacts the predictor's performance (Table 4, right), reporting results relative to the optimal configuration ($D = 16$). Extremely low dimensions ($D = 3$) severely underfit, causing a 12.96% drop in accuracy. Moderate dimensions ($D = 5$ or $D = 8$) partially capture model abilities, resulting in 1.58% and 2.81% decreases. Slightly larger dimensions ($D = 25$) perform comparably to the optimum, with only a 0.11% drop, indicating sufficient capacity without over-parameterization. Excessively

[1]Small LLM refers to Meta-Llama-3.1-70B for MMLU-Pro and GPT-4.1-Nano for ToolCall.
[2]Large LLM refers to Gemini-1.5-Pro for MMLU-Pro and o1 for ToolCall.
[4]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[5]https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[6]https://platform.openai.com/docs/models/text-embedding-3-large

Table 5: Effect of node label masking on predictor performance (relative drop ↓) and training cost (with savings ↑) on the ID dataset, without optimizer.

Node Masked (%)	Perf. (% ↓)	Training Cost ($) (Saving % ↑)
0	82.07 (-)	208.76 (-)
10	81.96 (↓0.11)	194.32 (6.91 ↑)
20	82.01 (↓0.06)	179.54 (13.99 ↑)
30	81.97 (↓0.10)	164.92 (21.00 ↑)
50	81.82 (↓0.25)	135.66 (35.03 ↑)
60	81.79 (↓0.28)	121.10 (42.00 ↑)
70	**81.48 (↓0.59)**	**106.23 (49.11 ↑)**

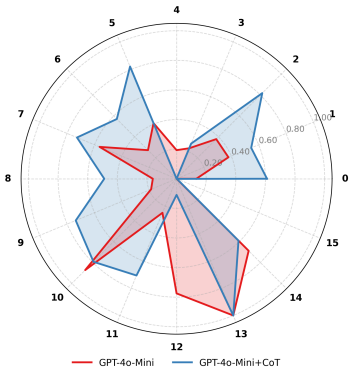


Figure 1: Comparison of GPT-4o-Mini and GPT-4o-Mini+CoT over 16 dimensions. The CoT variant improves performance on most dimensions, showing the benefit of explicit reasoning.

high dimensions ($D = 35$ or $D = 45$) introduce redundancy, causing 1.55% and 1.82% drops. Intermediate D offers the best balance between expressivity and generalization.

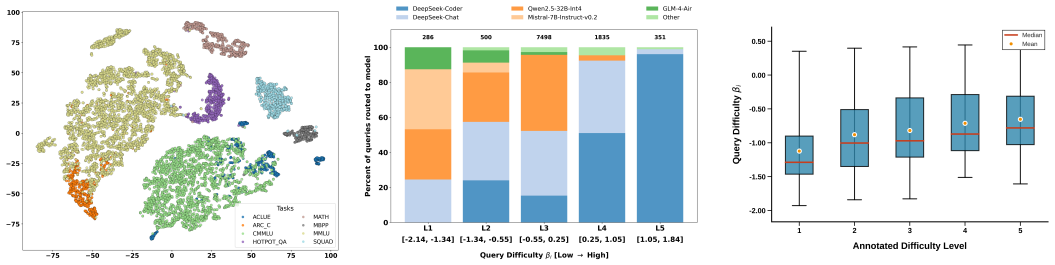
Semi-Supervised Cost-Efficient Training. Labeling all training nodes can be costly, since obtaining responses from commercial LLMs incurs significant expense. However, because GCNs naturally propagate label information across neighboring nodes, full supervision may be unnecessary. To quantify this, we simulate a semi-supervised setting by randomly masking a fraction of training node labels and report results in Table 5. The predictor demonstrates strong resilience to missing supervision. With 70% of nodes masked, the predictor achieves 81.48% accuracy, which is comparable to the state-of-the-art MIRT-Router without optimizer (81.17%, Table 12, Appendix A.7) and slightly below the fully supervised setting (82.07%), while reducing training cost by 49%. Intermediate masking levels (10%–30%) achieve proportional cost savings (6.91%–21%) with negligible performance degradation.

5.1 Interpretability of Pollinator

LLM Ability. We investigate ability differences across representative models within the same family using Pollinator. Figure 1 presents a comparison between GPT-4o-Mini and its chain-of-thought (CoT) augmented counterpart. The CoT-augmented model exhibits higher scores across most dimensions, demonstrating that explicit intermediate reasoning improves performance on reasoning-intensive tasks. As shown in Figure 4 in Appendix A.6, Llama3.1-405B-Instruct consistently matches or outperforms its smaller counterpart, Llama3.1-70B-Instruct. We evaluated this pair with a model ability dimension $D = 25$ to analyze trends in higher dimensions. Consistent with scaling laws ((Kaplan et al., 2020)), larger models demonstrate stronger performance. These observations are supported by Table 8 in Appendix A.5, showing that Llama3.1-405B-Instruct and GPT-4o-Mini+CoT outperform their smaller or base variants.

Query Difficulty. We evaluate whether Pollinator effectively estimates query difficulty using the MATH dataset, which includes human-annotated difficulty levels. As shown in Figure 2c, the estimated difficulty parameter β_i exhibits a clear monotonic increase across levels and aligns well with ground-truth annotations. Figure 5 in Appendix A.6 presents two representative queries with their predicted difficulty values, further highlighting the strong alignment between Pollinator's estimates and human-provided labels.

Routing Behavior Across Difficulty Levels. To understand how Pollinator balances performance and cost, we analyze routing decisions across queries stratified by difficulty (Figure 2b). Query difficulty, estimated via β_i, spans -2.14 to 1.84 and is divided into five linearly spaced bins (L1–L5). While models like DeepSeek-Coder, DeepSeek-Chat, and GPT-4o achieve the highest overall performance (Table 8 in Appendix A.5), Pollinator routes queries non-uniformly, leveraging difficulty-aware, cost-efficient allocation (see Table 7 in Appendix for model pricing). For



(a) (b) (c)

Figure 2: Visualization of interpretability analyses. **(a)** t-SNE projection of learned discrimination vectors α_i across in-domain datasets, showing task-specific clustering. **(b)** Routing distribution of models across query difficulty bins, where lightweight models handle easier queries while advanced models handle harder ones. **(c)** Predicted query difficulties (β_i) in the MATH dataset, grouped by human-annotated levels, showing monotonic alignment between model estimates and annotations.

easier queries (L1–L2), lightweight models such as Qwen2.5-32B-Int4, Mistral-7B-Instruct-v0.2, and GLM-4-Air dominate, minimizing cost while maintaining accuracy. In intermediate bins (L3–L4), routing is more heterogeneous, e.g., DeepSeek-Chat and Qwen2.5-32B-Int4. For the hardest queries (L5), top-tier models like DeepSeek-Coder are preferentially selected. POLLINATOR routes simple queries to inexpensive models and escalates complex ones to high-capacity models across the difficulty spectrum (Figure 2b).

Discrimination Vector Analysis. In POLLINATOR, the discrimination vector α_i encodes the skill requirements needed to solve a query. To assess whether these vectors capture task-level structure, we cluster queries based on their learned representations and project them into 2D using t-SNE (Figure 2a). Queries from the same dataset form cohesive clusters, showing that POLLINATOR effectively learns task-aligned skill representations. Some clusters partially overlap, reflecting shared skills: for example, ARC_C and MMLU overlap due to similar reasoning and problem-solving skills required, while CMMLU and ACLUE, the only two Chinese datasets among the eight, share the embedding space to some extent. In contrast, MATH (mathematics), SQuAD (reading comprehension), and MBPP (coding) form well-separated clusters, indicating that the learned vectors capture distinct task-specific skill requirements.

Alignment Between Discrimination Vectors and LLM Abilities. Table 11 in Appendix A.6 demonstrates that the discrimination vectors (α_i) learned by POLLINATOR align strongly with the ability profiles (θ_j) of individual LLMs. Comparing Qwen2.5-7B-Instruct with its math-specialized variant, Qwen2.5-Math-7B-Instruct, we find that queries, with higher mean routing probability, are directed to the general model on in-domain tasks overall, while math-focused queries (e.g., from MATH and GSM8K) are preferentially routed to the specialized model. Conversely, non-math queries from datasets such as MMLU and CommonsenseQA, are mostly routed to the general model. This indicates that the learned discrimination vectors (α_i) capture model-specific strengths and effectively guide query allocation.

6 CONCLUSION

We presented POLLINATOR, a data-efficient and online-serving-capable matchmaker for the intelligence marketplace. POLLINATOR combines a frugal GCN-based predictor with an IRT-head and an efficient dual-optimizer, reducing training cost by up to 49% while outperforming existing state-of-the-art predictors. Extensive experiments on real-world benchmarks, including BFCL-V3 and MMLU-Pro, demonstrate superior cost–performance trade-offs. Furthermore, detailed ablation studies and interpretability validate POLLINATOR's effectiveness for cost-efficient intelligence matchmaking. Future work will extend the framework to incorporate latency and volume constraints and explore adaptive dynamic graphs for evolving requests and producers.

REFERENCES

Kingma DP Ba J Adam et al. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 1412(6), 2014.

Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Xuanhui Wang. Personalized click shaping through lagrangian duality for online recommendation. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pp. 485–494, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314725. doi: 10.1145/2348283.2348350. URL https://doi.org/10.1145/2348283.2348350.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Kaidi Cao, Jiaxuan You, and Jure Leskovec. Relational multi-task learning: Modeling relations between data and tasks. *arXiv preprint arXiv:2303.07666*, 2023.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Shiming Chen, Ziming Hong, Guosen Xie, Qinmu Peng, Xinge You, Weiping Ding, and Ling Shao. Gndan: Graph navigated dual attention network for zero-shot learning. *IEEE transactions on neural networks and learning systems*, 35(4):4516–4529, 2022.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *The Thirteenth International Conference on Learning Representations*, 2024.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023.

Gauthier Guinet, Behrooz Omidvar-Tehrani, Anoop Deoras, and Laurent Callot. Automated evaluation of retrieval-augmented language models with task-specific exam generation. *arXiv preprint arXiv:2405.13622*, 2024.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Lars Erik Holmquist. Intelligence on tap: artificial intelligence as a new design material. *interactions*, 24(4):28–33, 2017.

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. *arXiv preprint arXiv:2403.12031*, 2024.

Jiaqi Huang, Chengxi Liu, Ziwei Wei, Yan Dong, Renze Zhang, Wanjun Zhou, Shiyue Zhang, Peiyi Lv, Peijie Wang, Zhihong Fan, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. *arXiv preprint arXiv:2306.09212*, 2024.

Qi Liu, Zheng Gong, Zhenya Huang, Chuanren Liu, Hengshu Zhu, Zhi Li, Enhong Chen, and Hui Xiong. Multi-dimensional ability diagnosis for machine learning algorithms. *Science China Information Sciences*, 67(12):229101, 2024.

Yang Liu, Alan Medlar, and Dorota Glowacka. What we evaluate when we evaluate recommender systems: Understanding recommender systems' performance using item response theory. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pp. 658–670, 2023.

Yunting Liu, Shreya Bhandari, and Zachary A Pardos. Leveraging llm respondents for item evaluation: A psychometric analysis. *British Journal of Educational Technology*, 56(3):1028–1052, 2025.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692*, 2023.

Kai Mei, Wujiang Xu, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing. *arXiv preprint arXiv:2502.20576*, 2025.

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 with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Ion Stoica, Joseph E. Gonzalez, et al. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Proceedings of the 2025 International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Mark D Reckase. 18 multidimensional item response theory. *Handbook of statistics*, 26:607–642, 2006.

Pedro Rodriguez, Joe Barrow, Alexander Miserlis Hoyle, John P Lalor, Robin Jia, and Jordan Boyd-Graber. Evaluation examples are not equally informative: How should that change nlp leaderboards? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4486–4503, 2021.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. Carrot: A cost aware rate optimal router. *arXiv preprint arXiv:2502.03261*, 2025.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. IRT-router: Effective and interpretable multi-LLM routing via item response theory. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15629–15644, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.761. URL https://aclanthology.org/2025.acl-long.761/.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Asterios Tsiourvas, Wei Sun, and Georgia Perakis. Causal llm routing: End-to-end regret minimization from observational data. *arXiv preprint arXiv:2505.16037*, 2025.

Fei Wang, Qi Liu, Enhong Chen, Zhenya Huang, Yuying Chen, Yu Yin, Zai Huang, and Shijin Wang. Neural cognitive diagnosis for intelligent education systems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 6153–6161, 2020.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. In *NeurIPS 2024 Track on Datasets and Benchmarks*, 2024. URL https://arxiv.org/abs/2406.01574.

David J Woodruff and Bradley A Hanson. Estimation of item response models using the em algorithm for finite mixtures. 1996.

Tian Xie, Bin Wang, and C-C Jay Kuo. Graphhop: An enhanced label propagation method for node classification. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):9287–9301, 2022.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Yixuan Zhang and Haonan Li. Can large language models comprehend ancient chinese? a preliminary test on aclue. *arXiv preprint arXiv:2310.09550*, 2023.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning compact representations of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Fs9EabmQrJ.

A APPENDIX

A.1 METHODOLOGY

The overall training and inference flows of POLLINATORare illustrated in Figure 3, showing the dual-tower encoder and IRT-based prediction head used to generate serving plans.

All the code and

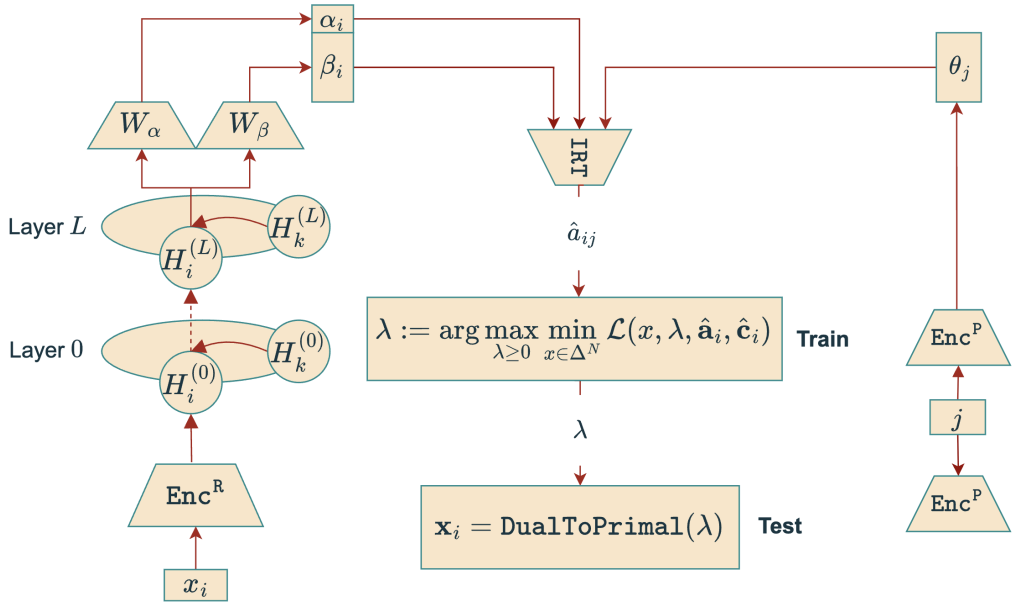


Figure 3: POLLINATOR: Train & Inference Flows. The left tower encodes request i with a GCN with L layers. The right tower encodes producer j. The bespoke IRT-based head combines the outputs of the two towers to generate ex ante predictions. In train flow, λ is the result of optimization on a held-out validation set, which, during inference, is used to compute the primal serving plan $\mathbf{x}_i$ via Algorithm 1.

A.2 DATASETS DETAILS

The datasets (Table 6) span a wide range of domains and task categories. In-domain datasets include reasoning, code, and QA tasks. Out-of-domain datasets test generalization to unseen tasks. Additional benchmarks, MMLU-Pro and BFCL V3 (Simple), evaluate more challenging reasoning problems and tool use. Each dataset lists the task type, evaluation metric, and train/test sizes.

A.3 CANDIDATE LLMS FOR VARIOUS DATASETS

For our routing experiments, we select a set of 20 representative LLMs as candidates for in-domain and out-of-domain datasets (see Table 7). The candidate LLMs, along with their input and output costs, for MMLU-Pro and BFCL-V3 (ToolCall) are reported in Tables 9 and 10, respectively.

A.4 PRICING OF CANDIDATE LLMS

We report both input and output token pricing ($/1M tokens) for all candidate models. Candidate LLMs exhibit drastic variation in pricing. Table 7 summarizes the base set of LLMs for in-domain and out-of-domain datasets, while Table 9 and Table 10 details the pricing of models used for MMLU-Pro and BFCL ToolCalling benchmarks.

Table 6: Details of in-domain, out-of-domain, and additional datasets used in our experiments.

In-domain				
Dataset	**Type**	**Evaluation Metric**	**Train Size**	**Test Size**
ACLUE (Zhang & Li, 2023)	Ancient Chinese	accuracy	1400	600
ARC_C (Clark et al., 2018)	Reasoning	accuracy	1400	600
CMMLU (Li et al., 2024)	Chinese Multitask	accuracy	7000	3000
Hotpot_QA (Yang et al., 2018)	Multi-Hop	EM	1400	600
MATH (Hendrycks et al., 2021)	Math	accuracy	1400	600
MBPP (Austin et al., 2021)	Code	pass@1	630	270
MMLU (Hendrycks et al., 2020)	Multitask	accuracy	9800	4200
SQuAD (Rajpurkar et al., 2018)	Reading Comprehension	f1	1400	600
Out-of-domain				
Dataset	**Task type**	**Evaluation Metric**	**Train Size**	**Test Size**
CEVAL (Huang et al., 2024)	Chinese Multitask	accuracy	-	1000
CommonsenseQA (Talmor et al., 2019)	Commonsense Reasoning	accuracy	-	1000
GSM8K (Cobbe et al., 2021)	Math	accuracy	-	1000
HumanEval (Chen et al., 2021)	Code	pass@1	-	160
Additional Datasets				
MMLU-Pro (Wang et al., 2024)	Multitask Reasoning	accuracy	9602	2430
BFCL V3 (Simple) (Patil et al., 2025)	Tool-Use / Function Calling	accuracy	125	75

Table 7: Pricing details of different LLMs ($/1M tokens) selected for In-Domain and Out-of-Domain Datasets.

LLM	**Input $/1M**	**Output $/1M**
DeepSeek-Chat	0.14	0.28
DeepSeek-Coder	0.14	0.28
Gemini-1.5-Flash	0.075	0.30
GLM-4-Air	0.137	0.137
GLM-4-Flash	0.0137	0.0137
GLM-4-Plus	6.85	6.85
GPT-4o	2.50	10.0
GPT-4o-Mini	0.15	0.60
GPT-4o-Mini+CoT	0.15	0.60
Llama3.1-8B-Instruct	0.10	0.20
Llama3.1-70B-Instruct	0.792	0.792
Llama3.1-405B-Instruct	3.15	3.15
Ministral-8B-Instruct-2410	0.10	0.20
Mistral-7B-Instruct-v0.2	0.10	0.20
Mixtral-8x7B-Instruct	0.54	0.54
Qwen2.5-32B-Instruct-GPTQ-Int4	0.10	0.20
Qwen2.5-7B-Instruct	0.10	0.20
Qwen2.5-72B-Instruct	1.08	1.08
Qwen2.5-Math-7B-Instruct	0.10	0.20
QwQ-32B-Preview	1.20	1.20

A.5 Average Performance of Models on ID data

Table 8 shows the average performance of 20 models on in-domain data. DeepSeek-Chat and DeepSeek-Coder achieve the highest performance, followed by large LLMs like Qwen2.5-72B-Instruct and GLM-4-Plus. Smaller or specialized models, such as Qwen2.5-Math-7B-Instruct, perform lower, reflecting their focus on niche tasks.

Table 8: Average performance of models on ID data (sorted in descending order).

Rank	Model	Performance
1	DeepSeek-Chat	0.8074
2	DeepSeek-Coder	0.8061
3	Qwen2.5-72B-Instruct	0.7997
4	GLM-4-Plus	0.7902
5	Qwen2.5-32B-Int4	0.7827
6	Llama3.1-405B-Instruct	0.7754
7	GPT-4o	0.7753
8	Gemini1.5-Flash	0.7290
9	GLM-4-Air	0.7200
10	GPT-4o-Mini+CoT	0.7171
11	Llama3.1-70B-Instruct	0.7111
12	GPT-4o-Mini	0.7067
13	Qwen2.5-7B-Instruct	0.6138
14	QwQ-32B-Preview	0.5973
15	GLM-4-Flash	0.5301
16	Ministral-8B-Instruct-2410	0.4872
17	Mixtral-8x7B-Instruct	0.3476
18	Llama3.1-8B-Instruct	0.3186
19	Mistral-7B-Instruct-v0.2	0.1042
20	Qwen2.5-Math-7B-Instruct	0.0980

Table 9: Pricing details of candidate LLMs selected for MMLU-Pro ($/1M tokens).

LLM	Input $/1M	Output $/1M
Claude-3.5-Sonnet	3.00	15.00
Gemini-1.5-Pro	1.25	5.00
Llama3.1-70B	0.60	0.60
Llama3.1-70B-Instruct	1.00	1.00
Llama3-70B	0.65	2.75
Llama3-70B-Instruct	0.59	0.79
Claude-3.5-Sonnet-(alt)	3.00	15.00
Claude-3.5-Sonnet-20241022	3.00	15.00
Claude-3.5-Haiku-20241022	0.80	4.00
Gemini-1.5-Flash	0.08	0.30
Gemini-2.0-Flash	0.15	0.60
GPT-4.1-Nano	0.10	0.40
GPT-4.1	2.00	8.00
GPT-4.1-Mini	0.40	1.60
o4-Mini	1.10	4.40

Table 10: Pricing details of candidate LLMs selected for BFCL Toolcalling ($/1M tokens).

LLM	Input $/1M	Output $/1M
GPT-4o	2.50	10.0
GPT-4o-Mini	0.15	0.60
o1	15.0	60.0
GPT-4.1-Nano	0.10	0.40
Gemini-1.5-Flash	0.08	0.30
Gemini-1.5-Pro	1.25	5.00
Gemini-2.0-Flash	0.15	0.60
GPT-4.1	2.00	8.00
GPT-4.1-Mini	0.40	1.60
o4-Mini	1.10	4.40

Table 11: Mean predicted routing probability of the general (Qwen2.5-7B-Instruct) vs. math-specialized (Qwen2.5-Math-7B-Instruct) models. Math queries are routed more often to the specialized model, while non-math queries favor the general model.

Task	Qwen2.5-7B-Instruct	Qwen2.5-Math-7B-Instruct
All ID Tasks	**0.60**	0.14
MATH	0.32	**0.70**
GSM8K	0.37	**0.44**
MMLU	**0.60**	0.04
CommonsenseQA	**0.74**	0.07

A.6 Additional Interpretability Results

Table 11 reports routing probabilities for Qwen2.5-7B-Instruct and its math-specialized variant. Math queries favor the specialized model, while general queries are routed to the general model, showing POLLINATOR's effective task-based model selection. Figure 4 highlights consistent performance improvements of Llama3.1-405B-Instruct over its smaller counterpart. Figure 5 illustrates example queries with predicted β_i, highlighting strong agreement with human labels.

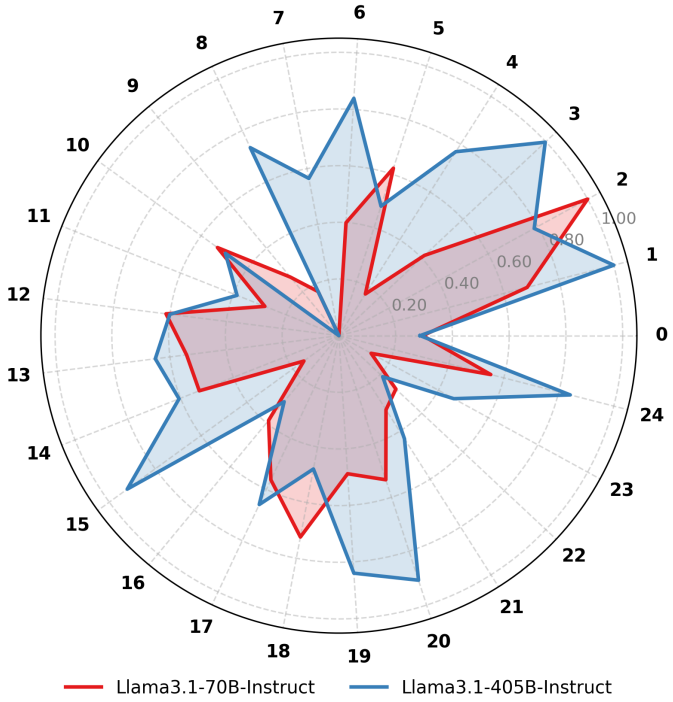


Figure 4: Estimated ability profiles (θ_j) over 25 dimensions. Comparison of Llama3.1-70B-Instruct and Llama3.1-405B-Instruct, showing consistent gains for the larger model.

A.7 Detailed Ablation Study of the Predictor

Table 12 presents a detailed ablation of our predictor (without the optimizer), analyzing the impact of ability dimension (θ), number of neighbors (k), edge-weighting, masking ratio, and embedder selection. Performance generally improves with increasing θ up to an optimal point. Across different configurations, our predictor consistently outperforms baseline methods, demonstrating robustness to design choices.

Type: Geometry
Level: Level 5

Problem: Points A, B, C, D, E, F, G, H, and K form a polygon such that each of the segments AB, BC, CD, DE, EF, FG, GH, and HK has length 4. All angles are right angles except at points D and F. In triangle DEF, which is an isosceles right triangle with legs DE = EF = 4, a perpendicular EM is drawn from E to DF. If EM = x, find x^2.

Difficulty: 0.7052

Type: Counting and Probability
Level: Level 1

Problem: Find the value of (3!)! / 3!.

Difficulty: -0.7126

Figure 5: Example queries with predicted b_i values compared to human labels, illustrating the close correspondence between predicted and true difficulty.

A.8 Candidate LLMs Profile Descriptions

Table 13 lists the candidate large LLMs used in our experiments for MMLU-Pro & BFCL-V3, along with their key profile descriptions.

Table 12: Ablation study of the predictor (without optimizer). Columns indicate ability (θ_j), dimension (D), number of neighbors (k), edge-weight, node label masking, request encoder ($\texttt{Enc}^{\texttt{R}}$), and performance. Bold values indicate the best configuration within each block.

Model	D	k	Edge-weight	Masking Ratio	$\texttt{Enc}^{\texttt{R}}$	Performance
Baselines						
Small LLM			✗	✗		48.70%
Large LLM			✗	✗		77.53%
KNN-Router		5	✗	✗		74.38%
MIRT-Router			✗	✗		81.17%
NIRT-Router			✗	✗		75.26%
POLLINATOR Ablation: D and k (no edge-weight)						
POLLINATOR	3	3	✗	✗	bert-base-uncased	71.04%
	5	3	✗	✗	bert-base-uncased	80.51%
	8	3	✗	✗	bert-base-uncased	79.26%
	16	3	✗	✗	bert-base-uncased	82.08%
	16	5	✗	✗	bert-base-uncased	81.73%
	16	10	✗	✗	bert-base-uncased	81.94%
	16	20	✗	✗	bert-base-uncased	82.03%
	25	3	✗	✗	bert-base-uncased	82.28%
	25	5	✗	✗	bert-base-uncased	82.24%
	25	10	✗	✗	bert-base-uncased	82.13%
	25	20	✗	✗	bert-base-uncased	81.53%
POLLINATOR with Edge-weight Vary D						
POLLINATOR	3	3	✓	✗	bert-base-uncased	71.43%
	5	3	✓	✗	bert-base-uncased	80.49%
	8	3	✓	✗	bert-base-uncased	79.26%
	16	3	✓	✗	bert-base-uncased	82.07%
	25	3	✓	✗	bert-base-uncased	81.96%
	35	3	✓	✗	bert-base-uncased	80.52%
	45	3	✓	✗	bert-base-uncased	81.25%
	3	5	✓	✗	bert-base-uncased	71.12%
	5	5	✓	✗	bert-base-uncased	80.27%
	8	5	✓	✗	bert-base-uncased	79.37%
	16	5	✓	✗	bert-base-uncased	81.90%
	25	5	✓	✗	bert-base-uncased	81.20%
	35	5	✓	✗	bert-base-uncased	80.53%
	45	5	✓	✗	bert-base-uncased	80.72%
POLLINATOR with Edge-weight						
POLLINATOR	16	3	✓	✗	bert-base-uncased	82.07%
	16	5	✓	✗	bert-base-uncased	81.92%
	16	10	✓	✗	bert-base-uncased	81.87%
	16	20	✓	✗	bert-base-uncased	82.12%
	25	3	✓	✗	bert-base-uncased	81.96%
	25	5	✓	✗	bert-base-uncased	82.20%
	25	10	✓	✗	bert-base-uncased	81.99%
	25	20	✓	✗	bert-base-uncased	81.68%
POLLINATOR with Node Masking						
POLLINATOR	16	3	✓	0.1	bert-base-uncased	81.96%
	16	3	✓	0.2	bert-base-uncased	82.01%
	16	3	✓	0.3	bert-base-uncased	81.97%
	16	3	✓	0.5	bert-base-uncased	81.82%
	25	5	✓	0.1	bert-base-uncased	81.79%
	25	5	✓	0.2	bert-base-uncased	81.89%
	25	5	✓	0.3	bert-base-uncased	82.12%
	25	5	✓	0.5	bert-base-uncased	81.68%
POLLINATOR with Different $\texttt{Enc}^{\texttt{R}}$						
POLLINATOR	16	3	✗	✗	all-MiniLM-L6-v2	80.86%
	25	3	✗	✗	all-MiniLM-L6-v2	80.74%
	16	3	✗	✗	all-mpnet-base-v2	81.70%
	25	3	✗	✗	all-mpnet-base-v2	82.37%
	25	3	✗	✗	text-embedding-3-large	81.97%
	16	3	✓	✗	all-mpnet-base-v2	81.70%
	16	3	✓	✗	all-MiniLM-L6-v2	80.75%
	16	3	✓	✗	text-embedding-3-large	80.15%
	25	5	✓	✗	all-mpnet-base-v2	82.60%
	25	5	✓	✗	all-MiniLM-L6-v2	80.74%
	25	5	✓	✗	text-embedding-3-large	81.91%

Table 13: List of candidate LLMs with their profile descriptions.

LLM Name	Profile Description
Gemini 2.0 Flash	Released on Dec 11, 2024 by Google DeepMind. Experimental version of Gemini 2.0 Flash, focusing on enhanced speed and performance. Features include a Multimodal Live API for real-time audio and video interactions, improved spatial understanding, native image and controllable text-to-speech with watermarking, and integrated tool use, including Google Search. Also introduces improved agentic capabilities and a new Google Gen AI SDK.
Gemini 1.5 Pro 002	Released on Sep 24, 2024 by Google DeepMind. Updated Gemini 1.5 Pro with a 2M token context window and up to 8,192 token outputs. Designed for diverse tasks via Google AI Studio and Vertex AI.
Meta-Llama-3.1-70B	Released on Jul 23, 2024 by Meta AI. A 70B parameter model pre-trained on 15T tokens from public sources, designed for advanced language understanding, coding, and reasoning.
Claude 3.5 Haiku	Released on Oct 22, 2024 by Anthropic. Optimized for efficiency and speed with a 200K token context window and 8,192 token outputs. Suitable for rapid-response tasks.
Meta-Llama-3.1-70B-Instruct	Released on Jul 23, 2024 by Meta AI. Instruction-tuned variant of Llama 3.1-70B, fine-tuned on public datasets and 10M+ human annotations to enhance instruction-following.
GPT-4.1 Nano	Released on Apr 14, 2025 by OpenAI. Compact GPT-4.1 version for on-device tasks with reduced compute needs while maintaining strong performance.
Gemini 1.5 Flash 002	Released on Sep 24, 2024 by Google DeepMind. Updated Gemini 1.5 Flash with a 1M token context window and up to 8,192 token outputs. Optimized for speed and cost-efficiency.
o4-Mini	Released on Aug 6, 2024 by OpenAI. A smaller GPT-4o variant with 1,047,476 token context window and 32,768 token outputs, offering efficiency while retaining robust performance.
Claude 3.5 Sonnet	Released on Oct 22, 2024 by Anthropic. Balances performance and efficiency with a 200K token context window and 8,192 token outputs. General-purpose model.
Claude 3.5 Sonnet	Released on Oct 22, 2024 by Anthropic. Part of the Claude 3.5 series, offering 200K context window, optimized for diverse tasks. Achieved 59.1% on GPQA Diamond benchmark.
Meta-Llama-3-70B	Released on Apr 18, 2024 by Meta AI. A 70B parameter model pre-trained on 15T tokens, designed for high-performance language tasks.
Meta-Llama-3-70B-Instruct	Released on Apr 18, 2024 by Meta AI. Instruction-tuned version of Llama 3-70B, aligned with user queries via public datasets and 10M+ annotations.
GPT-4.1	Released on Apr 14, 2025 by OpenAI. Enhanced GPT-4 variant with improved reasoning, coding, and agentic abilities.
GPT-4.1 Mini	Released on Apr 14, 2025 by OpenAI. Smaller GPT-4.1 variant, optimized for efficiency while maintaining strong task performance.
GPT-4.1-Nano	Released on April 14, 2025, by OpenAI. A compact version of the GPT-4.1 model, designed for on-device tasks with reduced computational requirements. Maintains strong performance across various benchmarks while being optimized for efficiency.
GPT-4o	GPT-4o: Released on November 20, 2023, by OpenAI. A large language model capable of handling complex tasks requiring deep understanding of language. Features include advanced reasoning capabilities, multimodal capabilities, and a context window of up to 128,000 tokens. Available through OpenAI's API.
GPT-4o-Mini	GPT-4o Mini: Released on November 20, 2023, by OpenAI. A smaller variant of the GPT-4o model, designed for efficiency while maintaining strong performance across various tasks. Optimized for applications requiring reduced computational resources.
o1	Released on August 16, 2024, by OpenAI. A large language model capable of handling complex tasks requiring deep understanding of language. Features include advanced reasoning capabilities, multimodal capabilities, and a context window of up to 128,000 tokens. Available through OpenAI's API.

B Additional Related Works

Item Response Theory. Item Response Theory (IRT) (Woodruff & Hanson, 1996) models the interaction between latent human ability and item difficulty via logistic functions, ensuring interpretability through monotonicity. Extensions such as MIRT (Reckase, 2006) and neural variants (e.g., NCDM (Wang et al., 2020)) capture richer interactions. Beyond education, IRT has been applied to model evaluation (Liu et al., 2024), recommendation (Liu et al., 2023), leaderboard ranking (Rodriguez et al., 2021), and LLM assessment (Guinet et al., 2024; Liu et al., 2025). We adopt IRT for its interpretability and proven effectiveness in human and machine assessment.

LLM Routers. LLM routing seeks to assign queries to the most suitable model for optimal accuracy–cost tradeoffs. Early works like FrugalGPT (Chen et al., 2023) and AutoMix (Aggarwal et al., 2023) use cascaded inference, while later methods train lightweight routers such as HybridLLM (Ding et al., 2024), RouteLLM (Ong et al., 2024), and Zooter (Lu et al., 2023). RouterDC (Chen et al., 2024) and KNN-based approaches (Hu et al., 2024) further reduce costs, and GraphRouter (Feng et al., 2024) leverages GNNs but depends on task priors. EmbedLLM (Zhuang et al., 2025) learns compact embeddings via matrix factorization to support routing at scale. Commercial systems like Martian[7] and Neutrino AI[8] demonstrate practical benefits, reporting major savings. Unlike these, our approach couples difficulty-aware estimation with online dual optimization, yielding interpretable and cost-efficient routing.

Graph-based Modeling. Graphs naturally capture relational structures (Fey et al., 2023; Cao et al., 2023; Chen et al., 2022). Classical methods like label propagation (Xie et al., 2022) leverage edges for transductive learning, while GNNs (Kipf, 2016; Hamilton et al., 2017) extend message passing to learn expressive representations. Recent work highlights their zero-/few-shot potential (Fey et al., 2023; Cao et al., 2023) in domains such as recommendation and social networks. Building on these advances, we employ GNNs to design the predictor of POLLINATOR [9]

[7] https://withmartian.com

[8] https://neutrinoapp.com

[9] All code and datasets for POLLINATOR are provided in the supplementary material.