ArtifactLinker: Linking Scientific Artifacts for Automatic State-of-the-art Discovery

Anonymous ACL submission

Abstract

Scientific artifacts such as models and benchmarks underpin machine learning research. With the rapid growth of repositories like HuggingFace, researchers now have access to millions of artifacts, yet a key challenge remains: how can we automatically discover the state-of-the-art (SOTA) model for a given benchmark by fully leveraging existing artifacts? We formalize this as **automatic SOTA discovery** by modeling HuggingFace as an artifact graph, where nodes are models/benchmarks and edges represent evaluations. We propose ARTIFACTLINKER, a two-stage framework: (1) prediction of promising unobserved model–benchmark links using Graph Neural Networks (GNNs) or graph-augmented Large Language Models (LLMs), and (2) verification via fully automatic, reproducible coding experiments with agents. We further introduce ARTIFACTBENCH with 2,977 models and 559 benchmarks to evaluate for both stages. Results show effective graph-based prediction and reliable end-to-end automatic verification of high-performing candidates.

1 Introduction

Scientific artifacts are the fundamental building blocks of research (Heumüller et al., 2020; Cooper et al., 2022; Johnson et al., 2019). Codebases on the GitHub platform, papers available on arXiv, models and benchmarks on the HuggingFace are all examples of such artifacts. Researchers engaged in doing reproducible and high-quality research share, interact with, and build upon these artifacts, releasing new versions to demonstrate progress (Marić et al., 2023; Lissa et al., 2020). In the machine learning community, a vast number of artifacts (>1M on HuggingFace) are produced by researchers working in different domains (Castaño et al., 2024; Ait et al., 2023; Laufer et al., 2025). This naturally raises an important question: *How can we leverage existing artifacts to enable automatic discovery?* Addressing this question would (1) allow us to utilize diverse types of artifacts better, and (2) promote scalable and automated scientific discovery based on existing resources. We focus on the HuggingFace community as a case study, since it is one of the largest and most active hubs of open-source machine learning artifacts and aims to make experiments more accessible and easy to run. With countless models, benchmarks, and libraries hosted on the platform, it provides an invaluable foundation for exploring automated discovery.

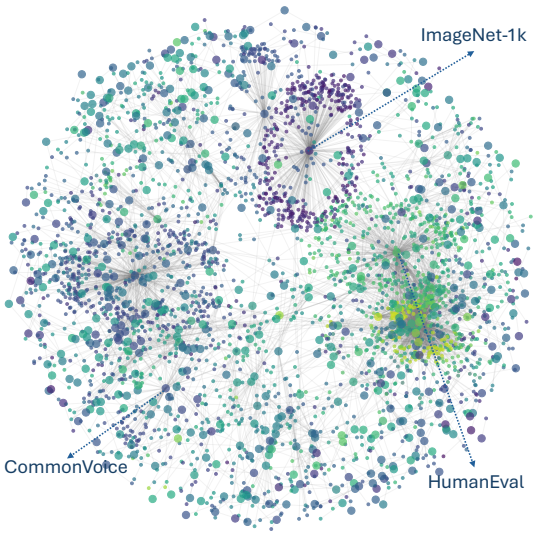

Figure 1: This shows the underlying graph of research artifacts from Huggingface Hub, used to build ARTIFACTBENCH, a new suite of tasks for developing automatic discovery agents. Large/small nodes represent models/datasets linked through evaluation, which we leverage for a new task called **SOTA discovery** (i.e., discovering novel links between existing artifacts).

Building an automatic discovery system on HuggingFace presents several challenges. First, the concept of "automatic discovery" itself is ill-defined (Beel et al., 2025; Kitano, 2021; Kramer et al., 2023)—what does it really mean for a system to conduct research autonomously and con-

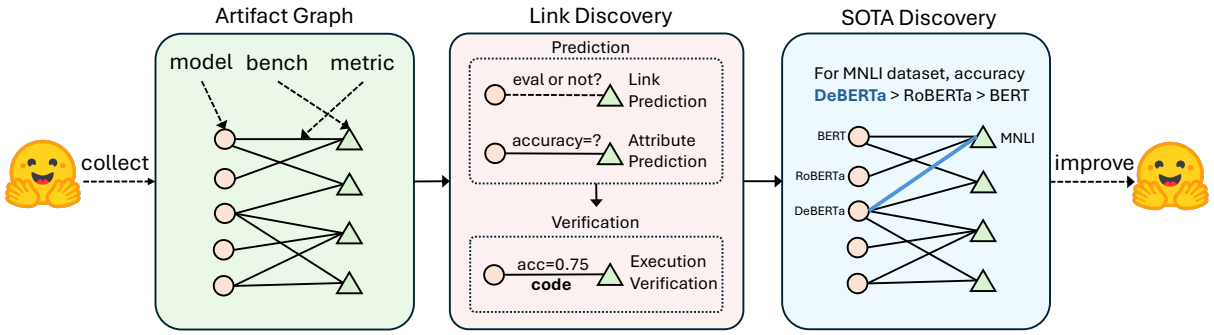

Figure 2: **Overview of ARTIFACTLINKER**. The overall automatic SOTA discovery process in ARTIFACTLINKER includes three main components: (1) artifact graph construction from HuggingFace (models, benchmarks, evaluation relationships); (2) link prediction and verification as discovery tasks; (3) state-of-the-art result filtering.

tribute to future applications? Second, although HuggingFace provides convenient access to models and datasets, building a fully automated and reproducible pipeline to verify predicted model performance is still difficult (Urbanowicz et al., 2022). The usability of models and benchmarks remains inconsistent, and many of them require specialized configurations to work properly and reproduce research works built on them, even a frustrating process for human researchers (Mu et al., 2025).

Thinking of HuggingFace as an artifact graph Our key observation is that the HuggingFace community can be naturally represented as a graph (Chen et al., 2025; Laufer et al., 2025) as shown in Figure 1 and Figure 2 (left), where models and benchmarks serve as nodes, and their relationships form the edges. This perspective is motivated by three characteristics of the platform: (1) it hosts a large collection of artifacts, including models and benchmarks; (2) many artifacts are high-quality and widely adopted; and (3) it encodes rich relational information—for example, a model trained on one dataset and evaluated on another with a reported F1 score. Such structured relationships are often difficult to extract directly from research papers. Prior work has largely treated HuggingFace as an information source for retrieval (Silva et al., 2025) or as an API hub (Shen et al., 2023). In contrast, we highlight its value for dynamic discovery. Rather than viewing it as a static data source, we aim to actively uncover new relationships—particularly model–benchmark interactions defined through evaluation metrics. These interactions naturally form the edges of an artifact graph.

Linking artifacts as automatic SOTA discovery Within this artifact graph, we define the task of automatic discovery as finding links with state-of-the-art evaluation results between artifacts (models and benchmarks). Concretely, our goal is to build an auto-discovery engine that leverages the existing artifact graph structures in the HuggingFace community and their existing relationships to propose new promising links.

Novel framework for automatic discovery To operationalize the linking problem as automatic discovery, we propose a novel framework with two stages (see Figure 2 (center)): (1) prediction and (2) verification. Given the vast number of artifacts in the graph and the high cost of verifying evaluation results, directly identifying the small subset of valuable links through execution alone is prohibitively costly. Our framework addresses this challenge by first performing prediction, where we use strong priors to filter out the majority of unlikely links—analogous to how experienced researchers prioritize promising directions. In the verification stage, we employ a coding agent to test and validate the predicted links, grounding hypotheses into real, reproducible results. This division of labor enables scalable automatic discovery while ensuring that results remain empirically verifiable.

Contributions Our work makes three key contributions: (1) we construct ARTIFACTBENCH, a challenging new benchmark for prediction, verification, and automatic discovery, (2) we design a two-stage framework named ARTIFACTLINKER for scalable link discovery, and (3) we demonstrate the practical value of this system through extensive evaluation and highlight current limitations that motivate further research on ARTIFACTBENCH.

2 Related Work

HuggingFace platform utilization HuggingFace has increasingly become a natural platform for studying automatic discovery. Prior work has largely relied on static analyses of its artifacts and relationships to characterize trends in machine learning development (Chen et al., 2025; Laufer

et al., 2025). Beyond serving as a repository, HuggingFace has been conceptualized in multiple ways: as a knowledge graph (Silva et al., 2025), an API hub (Shen et al., 2023), a model card aggregator (Yang et al., 2024), and even an evolutionary tree (Gao and Gao, 2023). Other studies have examined its community dynamics (Rahman et al., 2025; Castaño et al., 2023). In contrast, our work moves beyond static description and trend analysis and focuses on execution-based prediction/verification.

Large-scale prediction for accelerating discoveries Accelerating scientific discovery has been a major focus in domains such as drug discovery (Stokes et al., 2020; Serrano et al., 2024; Vișan and Neguț, 2024; You et al., 2022), materials science (Xie and Grossman, 2018; Butler et al., 2018), and molecular design (Segler et al., 2018), among others (Cheng et al., 2025). In these settings, experimental verification is prohibitively costly and time-consuming. In contrast, our work focuses on a more tractable class of automatic discovery tasks by leveraging the intrinsic linking structure of HuggingFace artifacts.

LLM-based coding agents for reproducible experimentation Prior work has explored free-form discovery with generating executable code from research ideas (Lu et al., 2024; Jansen et al., 2024, 2025), though evaluation remains challenging given the open-ended nature of such tasks. Other efforts have focused on reproducing experiments within specific codebases (Bogin et al., 2024; Starace et al., 2025; Kim et al., 2025; Seo et al., 2025; Siegel et al., 2024; Xiang et al., 2025), which is challenging due to the complexity of such codebases. In contrast, our tasks rely on reproducing a more concrete/grounded set of research artifacts.

3 Constructing an Artifact Graph from HuggingFace Hub

As the first stage of ARTIFACTLINKER, we first formally provide the definition of artifact graphs that we conduct link discovery on. Furthermore, we provide details about how we extract the artifact graph from the HuggingFace platform.

Definition of artifact graphs We define the artifact graph as an undirected heterogeneous bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V} = \mathcal{V}_m \cup \mathcal{V}_d$ consists of two disjoint types of nodes: **model nodes** $\mathcal{V}_m$ and **benchmark nodes** $\mathcal{V}_d$. The edge set $\mathcal{E} \subseteq \mathcal{V}_m \times \mathcal{V}_d \times \mathcal{K}$ encodes **evaluation relationships**, where each edge (u, v, k) indicates that a model $u \in \mathcal{V}_m$ has an evaluation result on a benchmark $v \in \mathcal{V}_d$ with score $k = \phi(u, v)$ defined by a metric function $\phi : \mathcal{V}_m \times \mathcal{V}_d \to \mathcal{K}$ (e.g., accuracy or F-score). Besides edges, each benchmark node is associated with attributes such as metadata (e.g., download counts) and task descriptions, while each model node is associated with metadata and specifications, including architecture, number of parameters, and configuration. Such rich information on both edges and nodes allows the graph to capture fine-grained performance relationships between models and benchmarks.

Data collection We construct edges in the artifact graph by extracting ground-truth evaluation metrics from HuggingFace model cards. Specifically, we parse the README files to identify model–dataset pairs along with their reported performance scores. Model and dataset names are then matched to their canonical entries on the HuggingFace platform to ensure consistency, yielding a clean set of evaluation edges. In total, this process produces $|\mathcal{E}| = 5{,}476$ perfectly matched evaluation records, which serve as edge attributes in the graph. For nodes, we collect both models and benchmarks. Each model is described by its model card, dataset card, and metadata fields (e.g., architecture, parameters, tags), while each dataset includes metadata such as domain, size, and license. To ensure relevance, we retain only nodes that participate in at least one evaluation edge (isolated artifacts).

Graph statistics The final artifact graph contains $|\mathcal{V}| = 3{,}536$ nodes, consisting of $|\mathcal{V}_m| = 2{,}977$ models and $|\mathcal{V}_d| = 559$ benchmarks, with $|\mathcal{E}| = 5{,}476$ edges in total. To quantify the community structure, we apply Louvain community detection (Blondel et al., 2008) and compute the modularity, obtaining $Q = 0.82$, which highlights the presence of clear sub-community structures under sparse connectivity. Figure 1 shows the visualization of the artifact graph structure. While benchmarks like ImageNet (Deng et al., 2009) and HumanEval (Chen, 2021) form expected hubs, such analysis indicates that edge concentration is limited and not solely focused on high-degree hubs.

4 Linking Scientific Artifacts for Automatic SOTA Discovery

To describe the overall pipeline of ARTIFACTLINKER, we first formalize the problem definition of automatic discovery using the artifact-graph formulation. We then introduce our scalable

solution, which addresses this problem via a two-stage *prediction–verification* framework.

4.1 Definition of Automatic SOTA Discovery

Building on the artifact graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ defined in the previous section, each observed edge $(m, d) \in \mathcal{E}$ is annotated with a performance score given by the metric function $\phi : \mathcal{V}_m \times \mathcal{V}_d \to \mathcal{K}$, with a realized score denoted as $\phi(m, d)$. The ultimate goal of automatic discovery is to identify a missing link $(m, d, k) \notin \mathcal{E}, m \in \mathcal{V}_m, d \in \mathcal{V}_d$ such that the predicted score under the same evaluation function $\phi(m, d)$ exceeds the best known performance on dataset d:

$$\phi(m, d) > \max_{(m', d) \in \mathcal{E}} \phi(m', d).$$

In other words, automatic SOTA discovery seeks model–benchmark pairs not yet connected in $\mathcal{G}$ but expected to advance the state of the art on d. The input of our proposed ARTIFACTLINKER framework is the artifact graph $\mathcal{G}$ while the target output is these state-of-the-art (m, d) pairs.

4.2 Scalable Link Discovery Framework

Because explicit model evaluation is computationally expensive, scalability requires pruning candidates before verification. We therefore adopt a two-stage framework with prediction and verification. In the prediction stage, we rank model–dataset pairs (m, d) using a score function $s(m, d)$ and select a candidate set $\mathcal{C}$ whose predicted scores approach or exceed the current SOTA. In the verification stage, candidates in $\mathcal{C}$ are executed and evaluated, producing the validated SOTA set $\mathcal{C}^*$.

4.2.1 Predicting via Score Function Modeling

The goal of prediction is not only to estimate $\hat{\phi}(m, d)$ for missing edges but also to rank all candidate links by their discovery potential. Concretely, each prediction model defines a scoring function $s : \mathcal{V}_m \times \mathcal{V}_d \to \mathbb{R}$, which assigns each unseen pair $(m, d) \notin \mathcal{E}$ a predicted performance. For each model the objective is to approximate the score function $\hat{\phi}(m, d)$ and produce rankings consistent with observed edges. Since all of our prediction models utilize graph information, we formulate them using a general GNN-style notation. Formally, each score function is defined as:

$$\mathbf{h}_v^{(k+1)} = \text{AGG}^{(k)}\big(\mathbf{h}_v^{(k)}, \{\mathbf{h}_u^{(k)}, e_{uv} : u \in \mathcal{N}(v)\}\big)$$
$$s(m, d) = \psi\big(\mathbf{h}_m^{(L)}, \mathbf{h}_d^{(L)}\big)$$

where $\mathbf{h}_v^{(k)}$ is a model's internal representation of node v after k iterations aggregated over all representations $\mathbf{h}_u^{(k)}$ constructed from neighboring nodes $u \in \mathcal{N}(v)$. The final score is $s(m, d)$ is then defined over the final node representations $\mathbf{h}_m^{(L)}$ and $\mathbf{h}_d^{(L)}$ transformed via some function ψ. As we detail below, different instantiations of AGG and ψ lead to different modeling strategies.

LLM modeling These models are based on ordinary prompt-based LLMs. Following the TextGNN formulation presented in Yu et al. (2024), we represent each $\mathbf{h}_v^{(k)}$ directly with text instead of embeddings. The aggregation function AGG is instantiated as a one-layer *concat* operator that serializes neighborhood information into a textual prompt:

$$\mathbf{h}_v^{(k+1)} = \text{CONCAT}\big(\mathbf{h}_v^{(k)}, \{\mathbf{h}_u^{(k)}, e_{uv} : u \in \mathcal{N}(v)\}\big)$$
$$s(m, d) = \text{LLM}\big(\mathbf{h}_m^{(L)}, \mathbf{h}_d^{(L)}\big)$$

Here, $\mathbf{h}_v^{(0)} = \mathbf{x}_v$ corresponds to the textual description of node v. The CONCAT function converts all neighbor attributes into natural language and concatenates them into a serialized context. The LLM then consumes these prompts for m and d, jointly reasoning over their neighborhoods to predict $s(m, d)$. We typically adopt a single TextGNN layer due to the limited context window size.

GNN-based modeling Alternatively, we instantiate AGG using standard message passing in Graph Neural Networks. In this setting, node features are initialized as $\mathbf{h}_v^{(0)} = \text{EMB}(\mathbf{x}_v)$ from the textual description $\mathbf{x}_v$, and representations are updated by:

$$\mathbf{h}_v^{(k+1)} = \sigma\big(W^{(k)}\mathbf{h}_v^{(k)} + \text{AGG}^{(k)}(\mathcal{N}(v))\big)$$
$$s(m, d) = \text{MLP}\big([\mathbf{h}_m^{(L)} \| \mathbf{h}_d^{(L)}]\big)$$

Here, $\text{AGG}^{(k)}(\cdot)$ can be implemented with classical GNN operators such as GATv2Conv (Brody et al., 2021), which applies dynamic attention weights to neighbor embeddings. The edge scoring function $s(m, d)$ is then computed by an MLP over the concatenation of the final node embeddings $\mathbf{h}_m^{(L)}$ and $\mathbf{h}_d^{(L)}$. In contrast to the LLM modeling above, this model is tuned using a training set of positive and negative links mined from the original artifact graph (see full details in Appendix. C.1)

Ranking with score function Given different modeling choices of $s(m, d)$, the prediction stage first produces a ranked list of all unseen pairs $(m, d) \notin \mathcal{E}$, ordered by their predicted score

$s(m, d)$. This ranking reflects the discovery potential of each candidate, with higher scores indicating stronger likelihood of advancing the state of the art. From this ranked list, we further define the set of promising candidates as

$$\mathcal{C} = \left\{ (m,d) \;\middle|\; \begin{array}{l} (m,d) \notin \mathcal{E},\ m \in \mathcal{V}_m,\ d \in \mathcal{V}_d, \\ s(m,d) \ \geq \ \max\limits_{(m',d)\in\mathcal{E}} (1-\delta)\phi(m',d) \end{array} \right\}$$

where $\delta(d) \in [0, 1]$ is a dataset-specific tolerance parameter and $\phi(m', d)$ is the existing evaluation score of model m' on dataset d under metric $\mu(d)$. Larger $\delta(d)$ indicates higher verification cost, but the greater possibility for finding the state-of-the-art performance. In other words, the score function $s(m, d)$ serves a dual purpose: it provides a global ranking over all candidates, and it defines a thresholded subset $\mathcal{C}$ to guide the verification stage by focusing on the most promising discoveries.

4.2.2 Verification via Multi-turn Tool Use

Given the candidate set $\mathcal{C}$ of manageable size, the goal of the verification stage is to execute each model–dataset pair and obtain concrete evaluation results. For each candidate (m, d), we perform verification using the CodeAct agentic workflow (Wang et al., 2024), which plans, generates, executes, and iteratively refines code in a multi-turn manner for scalable automatic benchmarking.

Specifically, the agent takes as input the model identifier m, the dataset identifier d, and a target evaluation metric $\mu(d)$ hooked with the dataset d (e.g., accuracy, F1, BLEU). We represent these inputs as descriptors m, d, and μ. To support reliable execution, the agent is equipped with HuggingFace-specific tools—such as `web_search`, `get_metadata`, and `get_readme`—which can be invoked at each turn to retrieve relevant model and dataset information. This tool-augmented agentic workflow enables reliable automatic evaluation:

$$s^*_\mu(m, d) \ = \ \textsc{CodeAct}\big(m, d, \mu(d)\big).$$

4.3 ARTIFACTLINKER Algorithm

Algorithm 1 presents the pipeline of ARTIFACTLINKER for automatic SOTA discovery. The algorithm proceeds in three stages: *(i)* compute the current SOTA *(ii)* perform attribute prediction over a fixed artifact graph(a transductive setting where nodes are observed and only missing links are predicted), with GNN-based methods enabling batch-parallel inference; and *(iii)* verify top candidates with a coding agent. Larger δ admits more candidates and thus increases computational overhead during verification. As a special case, ARTIFACTLINKER can also operate in a conditional mode, restricted to a specific dataset d or model m, reducing the need for full graph traversal. In an inductive setting, where a new model or benchmark is introduced, ARTIFACTLINKER can incorporate the new node via its attributes and apply the same predict-and-verify procedure to propose evaluations involving the unseen artifact.

Algorithm 1: SOTA discovery workflow

Input: Artifact graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; score model s; tolerance δ
Output: Validated SOTA discoveries $\mathcal{C}^*$

Precompute current SOTA (per dataset);
foreach $d \in \mathcal{V}_d$ **do**
  $b(d) \leftarrow \max_{(m',d)\in\mathcal{E}} \phi(m', d)$

Prediction: propose promising links;
$\hat{s} \leftarrow \textsc{AttrPredict}(s, \mathcal{G})$;
$\mathcal{C} \leftarrow \{(m, d) \notin \mathcal{E} \mid \hat{s}(m, d) \geq (1-\delta)b(d)\}$;

Verification: execute and validate;
$\mathcal{C}^* \leftarrow \emptyset$;
foreach $(m, d) \in \mathcal{C}$ **do**
  $s^*_\mu(m, d) \leftarrow \textsc{CodeAct}\big(m, d, \mu(d)\big)$
  if $s^*_\mu(m, d) > b(d)$ **then**
    $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{(m, d)\}$;

return $\mathcal{C}^*$;

5 Evaluating Automatic SOTA Discovery

To evaluate ARTIFACTLINKER, we design a mask-and-predict/verify protocol based on the artifact graph. Instead of running in the real world without ground-truths, we directly sample existing links $(m, d) \in \mathcal{E}$ and evaluate them on ARTIFACTBENCH by edge masking.

For evaluation, we consider two forms of missing links/masking. (i) *Edge masking*: a subset of observed edges $\mathcal{E}_{\text{mask}} \subset \mathcal{E}$ is hidden from the graph, requiring the model to recover their existence. (ii) *Attribute masking*: for another subset of edges, we keep the edge structure visible but hide their associated attributes (e.g., evaluation scores), requiring the model to predict/rank the missing values.

Prediction evaluation Given the masked graph $\mathcal{G} \setminus \mathcal{E}_{\text{mask}}$, we benchmark prediction performance with four tasks: (1) *Link prediction (edge masking)*: determine whether a masked edge $(m, d) \in \mathcal{E}_{\text{mask}}$

exists, i.e., $\hat{y}_{m,d} = \mathbf{1}[s(m,d) > \tau]$, where $s(m,d)$ is the predicted score function. (2) *Link ranking (edge masking)*: for each dataset d, rank all candidate models m by $s(m,d)$ and evaluate the rank of the true masked edges $(m,d) \in \mathcal{E}_{\text{mask}}$. (3) *Attribute prediction (attribute masking)*: for each edge (m,d) with hidden attributes, predict its score $\hat{\phi}(m,d)$. (4) *Attribute ranking (attribute masking)*: for each dataset d, rank all candidate models m by predicted attributes $\hat{\phi}(m,d)$ and compare this ordering against ground-truth scores.

Verification evaluation Finally, we define the *reproduction task* in the verification stage, which also corresponds to attribute masking. Here, the coding agent re-evaluates edges (m,d) with hidden attributes to measure consistency between predicted scores $s(m,d)$ and execution-based results $\phi^*(m,d)$. Formally, consistency is satisfied if $\left| s(m,d) - \phi^*(m,d) \right| \leq (1-\delta)\phi^*(m,d)$. We choose δ=0.1. This criterion checks whether prediction and execution agree up to an acceptable margin, ensuring faithfulness of the verification.

6 Experimental Settings

Prediction task settings We perform transductive edge-level splitting on the artifact graph, randomly partitioning edges into training, development, and test sets with a 70%–10%–20% ratio. The node set is fixed and shared across all splits, as our goal is to predict missing links in an existing graph. During training, message passing is restricted to the training edges, while development and test edges are held out for evaluation.

Prediction baseline settings Besides the LLM-based TextGNN settings and GNN-based settings, we include multiple baseline settings for comparison: (1) *metadata*, where we directly use metadata of the artifacts such as downloading times and metrics on edges as a feature for prediction; (2) *graph baselines*, where we utilize graph-related algorithm including Matrix Factorization (MF), Adamic-Adar (AA), Common-Neighborhood (CN) and more modern NeoGNN (Yun et al., 2021) baselines; (4) *LLM baselines*, where we only provide the model and benchmark information to an LLM (we use GPT-4o (OpenAI et al., 2024) and o3 (OpenAI, 2025b)) and ask them to predict and rank.

Verification task settings We construct the verification benchmark by selecting model–benchmark pairs from the artifact graph that include reported scores (e.g., F1, accuracy, BLEU, ROUGE). For each pair, we test whether the reported result can be reproduced through execution-based evaluation, given the model and benchmark specifications. We collect a subset of 93 model-dataset pairs and avoid datasets that are too expensive to run and models that are too large to download.

Method	Link Prediction		Link Ranking	
	F1 ↑	Acc ↑	R@5 ↑	P@5 ↑
Metadata	25.9	49.0	47.3	21.1
MF	29.5	40.5	37.6	19.1
CN	62.7	81.0	81.0	39.1
AA	67.2	84.5	81.0	39.1
GPT-4o	59.1	79.1	89.2	37.2
GPT-4o + graph	61.9	81.7	91.7	38.6
GPT-o3	59.1	80.4	90.3	38.3
GPT-o3 + graph	64.2	83.8	**92.6**	**40.2**
NeoGNN	63.8	91.1	43.3	16.0
GATv2Conv	**83.0**	**95.0**	71.9	33.4

Table 1: **Link prediction and ranking results.** For link prediction, we sample 5 negative samples for each datapoint with preferential attachment that chooses the highest degree nodes. For link ranking, we construct a benchmark-centric ranking task by randomly sampling 10 negative model candidates per benchmark. More details in Section §6.

Method	Attr Prediction		Attr Ranking
	MAE ↓	RMSE ↓	NDCG@1 ↑
Global mean	0.159	0.201	–
Local mean	0.107	0.156	–
GPT-4o	0.136	0.216	0.468
GPT-4o + graph	0.077	0.171	0.459
GPT-o3	0.168	0.260	0.433
GPT-o3 + graph	**0.060**	**0.126**	0.431
GATv2Conv	0.071	0.128	**0.519**

Table 2: **Attribute prediction and ranking results**. For attribute prediction, we focus on the 'accuracy', 'F1', 'BLEU', 'chrF', 'rouge' metrics for calculation in the test split to make sure the prediction range is unified. For attribute ranking, we build each ranking task for each dataset under one type of metric. More details in Section §6.

Verification baseline settings As baselines, we adopt single-turn code generation and multi-turn agentic CodeAct (Wang et al., 2024). Importantly, this baseline does not include HuggingFace-specific design choices, but simply uses a coding sandbox to conduct experiments automatically. Our method is built based on `smolagent` and, for a fair comparison, we use GPT-5.2 (OpenAI, 2025a) as the underlying backbone model, set max turn number to be 10, timeout for each turn to 900s.

7 Prediction Task Evaluation Results

Graph structure consistently improves LLM prediction Augmenting LLMs with graph neighborhood information yields consistent gains across

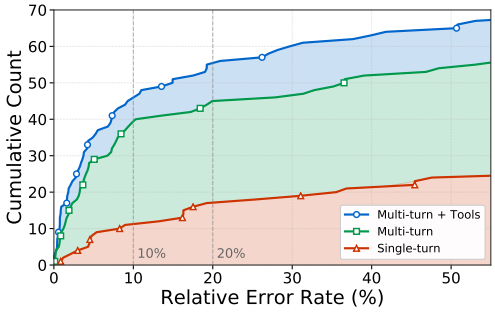

Figure 3: **Verification error rate distribution**. We calculate the cumulative count of datapoints with different error rates. Multi-turn and tool use mainly add more low-error-rate cases.

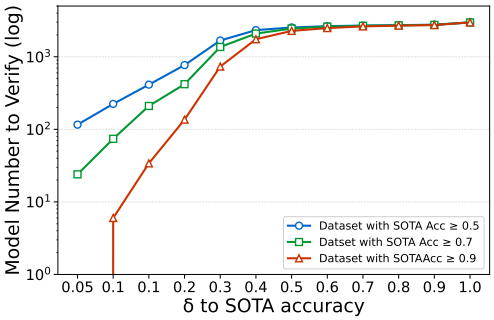

Figure 4: **Relationship between δ tolerance and the median number of candidate models to verify per dataset after prediction**. Given a dataset with known best accuracy, we use the GNN to predict accuracy for all 2,977 models and select candidates within $\delta\%$ of the existing SOTA scores.

tasks (Tables 1 and 2), especially for link and attribute prediction. For example, adding 1-hop context improves GPT-o3's link prediction F1 from 59.1 to 64.2 and reduces attribute MAE from 0.168 to 0.060. Ranking gains are smaller, likely due to strong semantic priors in LLMs. Under inductive splits with unseen models and datasets, performance degrades only mildly (e.g., GPT-4o+graph MAE increases from 0.07 to 0.08), indicating reasonable inductive robustness.

GNNs excel in transductive settings but generalize less inductively A lightweight 3-layer GATv2Conv with Voyage-3 embeddings achieves competitive or superior performance in the transductive setting as shown in Tables 1 and 2, matching GPT-4o+graph on attribute prediction and outperforming all baselines on link prediction. However, under inductive splits, GNN performance drops more substantially (MAE from 0.07 to 0.12), reflecting stronger dependence on observed graph structure compared to LLM-based methods.

Overall While current models are promising, significant headroom remains across all sub-tasks. These tasks go beyond metric evaluation and serve as proxies for scientific compatibility reasoning. Strong link and attribute prediction is a prerequisite for scalable automatic discovery, enabling agents to autonomously identify high-potential model–dataset combinations in the wild, highlighting the value of our task design.

8 Verification Task Evaluation Results

Multi-turn tool use enhances reproduction reliability As shown in Table 3, both multi-turn interactions and tool-use capabilities significantly improve the execution and reproduction rates of coding agents. Furthermore, Figure 3 illustrates that HuggingFace-specific tools effectively shift the distribution toward lower error rates, while the frequency of high-error cases remains unchanged. Specifically, enabling agents to search and fetch HuggingFace model cards and metadata yields a clear performance boost (3.2 points in completion rate and 4.3 points in success rate). We attribute this to the agent's ability to access model cards, which allows it to follow official code snippets and usage examples, thereby avoiding incorrect implementation. Despite these gains, several failure modes persist. The most common execution failures include missing result files (24%), zero-valued outputs (24%), and CUDA-related errors (12%). In addition, prompt sensitivity remains a key bottleneck: suboptimal evaluation prompts generated by our proposed coding agents can lead to unstable or degraded reproduced evaluation scores, particularly when evaluating LM candidates.

Overall Despite HuggingFace's robust execution infrastructure, current coding agents reproduce reported results only 45.2% of the time, indicating limited reliability. This reveals verification on the artifact graph as both a critical benchmark of agent capability and a largely unsolved problem, highlighting robust auto-evaluation agents as a key direction for future research.

Metric	Single-turn	Multi-turn	MT + tools
Completed (%)	29.0	77.4	**80.6**
Succeeded (%)	11.8	40.9	**45.2**
#turn / dp	1	4.2	4.7
#tool-use / dp	–	–	10.4

Table 3: **Verification performance with coding agents**. "`MT+tools`" represents our proposed multi-turn tool-use agent. "/dp" refers to per datapoint. We report task completion rate and success rates. The success rate indicates that reproduced results are in 10% difference with the reported results.

9 Discussion

Q1: What signals drive effective prediction in the artifact graph? Effective prediction arises

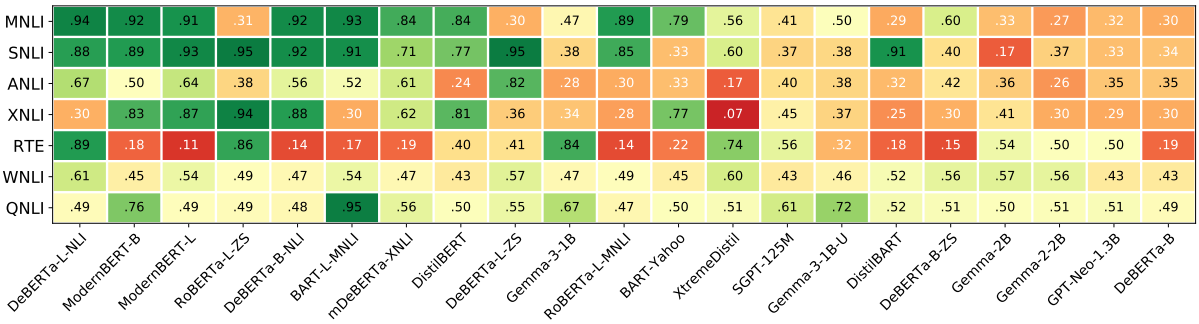

Figure 5: **Verification results on NLI tasks**. We show the accuracy verification results conducted by the ARTIFACTLINKER with 21 models and 7 NLI datasets. The results are conducted with ARTIFACTLINKER and might have false negatives existed due to limited abilities of the coding agent.

from the interaction between node attributes and graph structure. LLM-based predictors use attributes for analogical reasoning, while GNNs encode them via Voyage-3 embeddings and exploit dense connectivity through message passing. GNNs perform best on high-degree nodes, LLMs excel on medium-degree nodes (approximately 15–30), and both struggle on low-degree nodes. Randomizing text embeddings significantly degrades attribute prediction RMSE. All predictions rely on LLM-summarized model cards with ground-truth labels removed to avoid leakage.

Q2: How does prediction reduce verification costs? We analyze how many models need to be verified per dataset after prediction-based filtering. For datasets with SOTA accuracy $\geq$0.7, we count models predicted within a certain ratio of the ground-truth SOTA in Figure 4. At 5% tolerance, the median is 24 candidate models—a 99% reduction from the total. At 20% tolerance, the median rises to 419. This suggests 20-80 experiments typically suffice to identify near-optimal models.

10 End-to-end Case Study

To demonstrate the effectiveness of ARTIFACTLINKER when used to uncover real unseen links, we conduct a study on the classic task of Natural Language Inference (NLI). We consider seven representative benchmarks: MNLI (Williams et al., 2018), SNLI (Bowman et al., 2015), ANLI (Nie et al., 2020), XNLI (Conneau et al., 2018), RTE (Bentivogli et al., 2009), WNLI, and QNLI (Wang et al., 2018). ARTIFACTLINKER first predicts promising model–dataset pairs over the artifact graph, after which we select 21 relatively small models for fully automated coding-based verification. Results are shown in Figure 5.

SOTA discovery All evaluations in Figure 5 are conducted fully automatically without human intervention. While some scores may be imperfect due to limitations of the coding agent, the results nonetheless reveal several meaningful discoveries. Notably, ARTIFACTLINKER identifies a new state-of-the-art result on SNLI: deberta-v3-large-zeroshot-v2.0[1] , which does not report official SNLI results, achieves an accuracy of 0.95. We first verify that this score is consistent with the state-of-the-art estimates derived from our artifact graph. Furthermore, we confirm that it is similar to the best result reported on the SNLI leaderboard [2]. We also include reproducible scripts in Appendix. E.

Research insight Beyond SOTA discovery, we observe strong performance correlation between MNLI and SNLI across multiple models, strong QNLI performance from BART-large-mnli[3], and unexpectedly strong RTE results from Gemma-3-1B. Overall, this case study demonstrates ARTIFACTLINKER 's ability to efficiently discover SOTA-level results and uncover research insights not explicitly reported in HF, highlighting its potential as a scalable tool for automatic research.

11 Conclusion

We introduce ARTIFACTLINKER, a framework that enables automatic SOTA discovery on HuggingFace by modeling models and benchmarks as an artifact graph. Using a two-stage predict-and-verify approach—graph-based candidate prediction followed by execution-based verification. We further propose ARTIFACTBENCH to evaluate both stages, and show that ARTIFACTLINKER can discover high-performing links and reproduce many benchmark results automatically, pointing toward scalable and continual scientific discovery.

[1] https://huggingface.co/MoritzLaurer/deberta-v3-large-zeroshot-v2.0

[2] https://nlp.stanford.edu/projects/snli/

[3] https://huggingface.co/facebook/bart-large-mnli

Limitations

Computational considerations in verification While our two-stage framework effectively reduces the search space through candidate filtering in the prediction phase, the verification stage requires actual code execution for validation. For certain large-scale experiments or resource-intensive benchmarks, full reproduction may require non-trivial computational costs depending on available hardware resources. We note that our current implementation successfully handles the majority of common benchmarks, though scaling to extremely high-throughput scenarios across diverse hardware environments remains an interesting direction for future optimization.

Evaluation metric coverage Our framework currently emphasizes objective, quantitative metrics such as Accuracy, F1, and Exact Match, which are widely reported in model documentation and amenable to automatic extraction. These metrics cover a substantial portion of commonly used evaluation schemes in NLP. While our approach could potentially be extended to incorporate tasks involving human evaluation or qualitative assessment, we leave the integration of such subjective metrics as a natural extension for future work, as they would require additional methodological considerations for automated processing.

Ethics Consideration

Our work focuses on building an automatic discovery framework over open-source artifacts available on HuggingFace. All experiments are conducted exclusively on publicly released models and benchmarks that are freely accessible to the research community. We do not introduce any new human or sensitive data, nor do we attempt to deanonymize or misuse existing artifacts. The goal of our framework is to advance automated, reproducible, and scalable scientific discovery, and to help researchers more efficiently identify promising model–benchmark interactions. Nevertheless, we acknowledge that automated benchmarking may propagate existing biases and limitations present in the underlying models and datasets. To mitigate this, we emphasize transparency in data collection and reproducibility in our verification pipeline. We encourage the community to view our work as a step toward building more reliable and responsible auto-discovery systems, rather than a replacement for human oversight.

References

Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2023. On the suitability of hugging face hub for empirical studies. *ArXiv*, abs/2307.14841.

Joeran Beel, Min-Yen Kan, and Moritz Baumgart. 2025. Evaluating sakana's ai scientist for autonomous research: Wishful thinking or an emerging reality towards 'artificial research intelligence' (ari)? *ArXiv*, abs/2502.14297.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. *TAC*, 7(8):1.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. 2024. Super: Evaluating agents on setting up and executing tasks from research repositories. *Proceedings of EMNLP*.

Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 632–642.

Shaked Brody, Uri Alon, and Eran Yahav. 2021. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

Keith T Butler, Daniel W Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. 2018. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555.

Joel Castaño, Rafael Cabañas, Antonio Salmer'on, David Lo, and Silverio Mart'inez-Fern'andez. 2024. How do machine learning models change? *ArXiv*, abs/2411.09645.

Joel Castaño, Silverio Martínez-Fernández, Xavier Franch, and Justus Bogner. 2023. Analyzing the evolution and maintenance of ml models on hugging face. *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pages 607–618.

Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Qiaosheng Chen, Kaijia Huang, Xiaofang Zhou, Weiqing Luo, Yuanning Cui, and Gong Cheng. 2025. Benchmarking recommendation, classification, and tracing based on hugging face knowledge graph. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Junyan Cheng, Peter Clark, and Kyle Richardson. 2025. Language modeling by language models. *Proceedings of NeurIPS*.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. Xnli: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2475–2485.

Ned Cooper, Tiffanie N. Horne, Gillian R. Hayes, Courtney Heldreth, Michal Lahav, Jess Holbrook, and Lauren Wilcox. 2022. A systematic review and thematic analysis of community-collaborative approaches to computing research. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Sarah Gao and Andrew Gao. 2023. On the origin of llms: An evolutionary tree and graph for 15, 821 large language models. *ArXiv*, abs/2307.09793.

R. Heumüller, Sebastian Nielebock, J. Krüger, and F. Ortmeier. 2020. Publish or perish, but do not forget your software artifacts. *Empirical Software Engineering*, 25:4585 – 4616.

Peter Alexander Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. 2024. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. *ArXiv*, abs/2406.06769.

Peter Alexander Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi, Bodhisattwa Prasad Majumder, D. S. Weld, and Peter Clark. 2025. Codescientist: End-to-end semi-automated scientific discovery with code-based experimentation. *ArXiv*, abs/2503.22708.

Seth Johnson, F. Samsel, G. Abram, Daniel L. Olson, Andrew J. Solis, Bridger Herman, P. Wolfram, C. Lenglet, and Daniel F. Keefe. 2019. Artifact-based rendering: Harnessing natural and traditional visual media for more expressive and engaging 3d visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 26:492–502.

Gyeongwon James Kim, Alex Wilf, Louis philippe Morency, and Daniel Fried. 2025. From reproduction to replication: Evaluating research agents with progressive code masking. *ArXiv*, abs/2506.19724.

H. Kitano. 2021. Nobel turing challenge: creating the engine for scientific discovery. *NPJ Systems Biology and Applications*, 7.

Stefan Kramer, Mattia Cerrato, S. Džeroski, and R. King. 2023. Automated scientific discovery: From equation discovery to autonomous discovery systems. *ArXiv*, abs/2305.02251.

Benjamin Laufer, Hamidah Oderinwale, and Jon Kleinberg. 2025. Anatomy of a machine learning ecosystem: 2 million models on hugging face. *ArXiv*, abs/2508.06811.

C. J. Lissa, A. Brandmaier, Loek Brinkman, Anna-Lena Lamprecht, Aaron Peikert, Marijn E. Struiksma, and Barbara M. I. Vreede. 2020. Worcs: A workflow for open reproducible code in science. *Data Sci.*, 4:29–49.

Chris Lu, Cong Lu, R. T. Lange, J. Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *ArXiv*, abs/2408.06292.

T. Marić, Dennis Gläser, Jan-Patrick Lehr, Ioannis Papagiannidis, B. Lambie, Christian H. Bischof, and Dieter Bothe. 2023. A pragmatic workflow for research software engineering in computational science. *ArXiv*, abs/2310.00960.

Yanzhou Mu, Rong Wang, Juan Zhai, Chunrong Fang, Xiang Chen, Jiacong Wu, An Guo, Jiawei Shen, Bingzhuo Li, and Zhenyu Chen. 2025. Understanding llm-centric challenges for deep learning frameworks: An empirical analysis. In *unknown*.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial nli: A new benchmark for natural language understanding. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4885–4901.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

OpenAI. 2025a. Introducing OpenAI gpt-5.2. Accessed: 2026-01-06.

OpenAI. 2025b. Introducing OpenAI o3 and o4-mini. Accessed: 2026-01-06.

Mohammad Shahedur Rahman, Peng Gao, and Yuede Ji. 2025. Hugginggraph: Understanding the supply chain of llm ecosystem. *ArXiv*, abs/2507.14240.

Marwin HS Segler, Mike Preuss, and Mark P Waller. 2018. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2025. Paper2code: Automating code generation from scientific papers in machine learning. *ArXiv*, abs/2504.17192.

D. Serrano, F. C. Luciano, B. J. Anaya, Baris Ongoren, Aytug Kara, Gracia Molina, Bianca I Ramirez, Sergio A Sánchez-Guirales, Jesus A. Simon, Greta Tomietto, Chrysi Rapti, Helga K. Ruiz, Satyavati Rawat, Dinesh Kumar, and A. Lalatsa. 2024. Artificial intelligence (ai) applications in drug discovery and drug delivery: Revolutionizing personalized medicine. *Pharmaceutics*, 16.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Y. Zhuang. 2023. Taskbench: Benchmarking large language models for task automation. *ArXiv*, abs/2311.18760.

Zachary S. Siegel, Sayash Kapoor, Nitya Nagdir, Benedikt Stroebl, and Arvind Narayanan. 2024. Core-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *Trans. Mach. Learn. Res.*, 2024.

Kanishka Silva, Marcel R. Ackermann, Heike Fliegl, G. Gesese, Fidan Limani, Philipp Mayr, Peter Mutschke, A. Oelen, Muhammad Asif Suryani, Sharmila Upadhyaya, Benjamin Zapilko, Harald Sack, and Stefan Dietze. 2025. Research knowledge graphs in nfdi4datascience: Key activities, achievements, and future directions. *ArXiv*, abs/2508.02300.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, and 1 others. 2025. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, and 1 others. 2020. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.

R. Urbanowicz, Robert F. Zhang, Yuhan Cui, and Pranshu Suri. 2022. Streamline: A simple, transparent, end-to-end automated machine learning pipeline facilitating data analysis and algorithm comparison. In *Genetic Programming Theory and Practice*.

A. Vișan and I. Negut. 2024. Integrating artificial intelligence for drug discovery in the context of revolutionizing drug delivery. *Life*, 14.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pages 353–355.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long papers)*, pages 1112–1122.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. 2025. Scireplicate-bench: Benchmarking llms in agent-driven algorithmic reproduction from research papers. *ArXiv*, abs/2504.00255.

Tian Xie and Jeffrey C Grossman. 2018. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301.

Xinyu Yang, Weixin Liang, and James Zou. 2024. Navigating dataset documentations in ai: A large-scale analysis of dataset cards on hugging face. *ArXiv*, abs/2401.13822.

Yujie You, Xin Lai, Ying Pan, Huiru Zheng, J. Vera, Suran Liu, Senyi Deng, and Le Zhang. 2022. Artificial intelligence in cancer target identification and drug discovery. *Signal Transduction and Targeted Therapy*, 7.

Haofei Yu, Zhaochen Hong, Zirui Cheng, Kunlun Zhu, Keyang Xuan, Jinwei Yao, Tao Feng, and Jiaxuan You. 2024. Researchtown: Simulator of human research community. *arXiv preprint arXiv:2412.17767*.

Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. 2021. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems*, 34:13683–13694.

A Potential Risks

Metadata Quality Considerations. Our prediction stage uses metadata from the HuggingFace community, including model descriptions, architectural specifications, and training configurations. As with any crowdsourced platform, there may occasionally be instances of incomplete or imprecise documentation that could affect initial ranking predictions.

Performance-Oriented Discovery Scope. ARTIFACTLINKER is designed to identify high-performing model-benchmark combinations based on standard evaluation metrics. As with any automated performance discovery tool, users should exercise standard research practices by conducting appropriate safety and ethical assessments before deploying discovered models in production environments, particularly for applications involving user-facing content generation. We view our framework as a research tool that augments human decision-making rather than replacing it. Users retain full responsibility for evaluating whether discovered models meet the safety, fairness, and ethical requirements of their specific use cases, and we encourage comprehensive evaluation beyond performance metrics alone.

B Scientific Artifacts

B.1 Data license

All data in ARTIFACTBENCH are derived from the HuggingFace Hub, which hosts open-source models and benchmarks under various public licenses (*e.g.*, Apache 2.0, MIT, and CC-BY). We plan to release ARTIFACTBENCH under the Open Database License (ODbL), which permits use, sharing, and modification of the data while requiring proper attribution and that any derivative works remain openly available under the same license.

B.2 Model license

Our work relies on multiple foundation models, including `chatgpt-4o-latest` and `o3-2025-04-16`. Specifically, we access `chatgpt-4o-latest` and texttto3-2025-04-16 via the official OpenAI API. We utilize the official inference API provided by VoyageAI to use voyage-3.

The `chatgpt-4o-latest`, `o3-2025-04-16`, and voyage-3 models are closed-source and operate under proprietary licenses. We use them only for academic, and non-commercial purposes and ensure all inputs come from publicly available data, complying with their usage restrictions. We make no modifications to these models and use them as-is via their public APIs.

B.3 Data Usage

All data used in this study, including model specifications, benchmark descriptions, and evaluation results, were collected from the publicly available HuggingFace Hub. Our data collection focuses exclusively on scientific artifacts (models, datasets, and benchmarks) and does not include any personally identifiable information. All collected data are derived from publicly released resources that are freely accessible to the research community, in compliance with the HuggingFace platform's Terms of Service.

C Experimental Details

C.1 GNN-based Modeling and Training Details

In this section, we describe the architecture of our GNN-based model and provide details on its training setup for both link prediction and attribute regression tasks.

C.2 Model Architecture

Our link prediction model, `GNNLinkPredictor`, combines a graph neural network encoder with a flexible edge-scoring decoder. The encoder is based on GATv2 and stacks multiple convolutional layers with `GraphNorm`, `PReLU` activations, and feature dropout. Residual connections are added when input and output dimensions match, and self-loops as well as edge dropout can be applied for stability. To integrate information across layers, we adopt Jumping Knowledge (JK), supporting concatenation with projection, element-wise max pooling, or using the last layer only. The resulting node embeddings are passed into the edge decoder (`EdgePredictor`), which supports various scoring functions, including dot product, cosine similarity, bilinear transformation, linear projection on concatenated features, and a two-layer MLP. This design balances expressiveness and flexibility: the encoder captures rich multi-hop structure, while the decoder adapts to diverse relational patterns.

C.3 Training for Link Prediction

For binary edge classification, we train our GNN model over positive and negative edges sampled from the artifact graph, using pre-computed node

embeddings as input. We optimize with AdamW (learning rate 5×10^{-3}, weight decay 10^{-4}) and use a ReduceLROnPlateau scheduler (patience 10, decay factor 0.8, minimum learning rate 10^{-6}). Training runs for up to 300 epochs with early stopping (patience 40) based on validation AUC, and class imbalance is corrected by weighting the binary cross-entropy loss with $\frac{N_{\text{neg}}}{N_{\text{pos}}}$. Mixed-precision training is enabled on GPU. We evaluate every 50 epochs, select the best checkpoint by validation AUC, and report accuracy, precision, recall, F1, ROC-AUC, and average precision on the test set. Unless otherwise noted, we use 64 hidden dimensions, 3 layers, 3 heads, dropout 0.2, and seed 42.

C.4 Training for Attribute Regression

For continuous edge-attribute prediction, we train on positive edges with metric values, normalizing values greater than 1 to $[0, 1]$. Node embeddings are pre-computed and message passing uses the training graph's edges. The model regresses edge scores directly in logit space: ground-truth targets are transformed via $\text{logit}(y) = \log \frac{y}{1-y}$, and MSE is computed against the raw decoder logits. This stabilizes training near the boundaries. We use Adam (learning rate 0.005, weight decay 10^{-5}), Xavier-uniform initialization for linear layers, dropout 0.2, hidden size 128, 3 layers, and 8 heads (GATv2 backbone). Training runs up to 500 epochs, with frequent evaluation early (every 10 epochs for the first 50, then every 25), and the best checkpoint is selected by lowest validation MSE. At evaluation, logits are clipped to $[-10, 10]$ and mapped through sigmoid to report MSE, MAE, RMSE, R^2, and MAPE.

C.5 Model Size and Budget

For the close-sourced models, the model size is unknown. The GNN model we used is within 100 MB. The budget for conducting prediction/ranking/verification is <$1000.

D The Use of Large Language Models (LLMs)

We used ChatGPT as a writing assistant to help us write part of the paper. Additionally, we utilize the power of CodePilot to help us code faster. However, all the AI-generated writing and coding components are manually checked and modified. There is no full AI-generated content in the paper.

E SOTA Discovery Details

We provide the discovered state-of-the-art evaluation code here:

Listing 1: Automatic SNLI evaluation for SOTA verification.

```
import json
import random
import torch
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoConfig,
    AutoModelForSequenceClassification
)

MODEL_ID =
    "MoritzLaurer/deberta-v3-large-zeroshot
-v2.0"
DATASET_ID = "stanfordnlp/snli"
SPLIT = "validation"
MAX_SAMPLES = 20000
BATCH_SIZE = 16
MAX_LENGTH = 256
SEED = 42

random.seed(SEED)
torch.manual_seed(SEED)

dataset = load_dataset(DATASET_ID,
    split=SPLIT)

def is_valid(example):
    return (
        example["label"] is not None
        and example["label"] != -1
        and example["premise"]
        and example["hypothesis"]
    )

valid_indices = [i for i in
    range(len(dataset)) if
    is_valid(dataset[i])]
random.shuffle(valid_indices)
dataset =
    dataset.select(valid_indices[:MAX_SAMPLE
S])

tokenizer =
    AutoTokenizer.from_pretrained(MODEL_ID,
    use_fast=True)
config =
    AutoConfig.from_pretrained(MODEL_ID)
model = AutoModelForSequenceClassification.
from_pretrained(MODEL_ID)

device = torch.device("cuda" if
    torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# Determine model label mapping
if getattr(config, "label2id", None) and
    len(config.label2id) > 0:
    label2id = {k.lower(): int(v) for k, v
    in config.label2id.items()}
else:
    label2id = {v.lower(): int(k) for k, v
```

```
    in config.id2label.items()}

entailment_id = label2id["entailment"]
not_entailment_id =
    label2id["not_entailment"]

snli_labels =
    dataset.features["label"].names
snli_entailment_id =
    snli_labels.index("entailment")

correct, total = 0, 0

for i in range(0, len(dataset), BATCH_SIZE):
    batch = dataset[i:i+BATCH_SIZE]

    enc = tokenizer(
        batch["premise"],
        batch["hypothesis"],
        padding=True,
        truncation=True,
        max_length=MAX_LENGTH,
        return_tensors="pt"
    )
    enc = {k: v.to(device) for k, v in
    enc.items()}

    labels = [
        entailment_id if y ==
    snli_entailment_id else
    not_entailment_id
        for y in batch["label"]
    ]

    with torch.no_grad():
        preds =
    model(**enc).logits.argmax(dim=-1).cpu().
        tolist()

    for p, y in zip(preds, labels):
        correct += int(p == y)
        total += 1

accuracy = correct / total
print("N␣=", total)
print("Accuracy␣=", accuracy)

with open("results.json", "w") as f:
    json.dump({"accuracy": accuracy}, f)
```

F LLM-based Modeling Details

We directly rely on in-context learning for LLM-based modeling without training. We provide examples of our prompts for different tasks below:

Listing 2: LLM with graph link prediction example

```
Given a machine learning model named
    'tensorblock/bloomz-3b-GGUF' and a
    dataset named 'SetFit/rte'.

More information about this model: The
    model 'bloomz-3b1' is a multilingual
    text-generation model based on the
    BLOOM architecture. It is fine-tuned on
    a variety of datasets such as the
    BigScience xP3, which contributes to
    its ability to handle multiple
    languages and tasks. The model supports
    a wide range of languages including
    English, Spanish, French, Chinese, and
    many others, making it highly versatile
    for global applications. It is also
    capable of understanding and generating
    programming code in several languages,
    like C++, Java, Python, and more. This
    flexibility is reflected in its usage
    across diverse tasks, from sentiment
    analysis and question answering to
    natural language inference and program
    synthesis. The model operates under the
    bigscience-bloom-rail-1.0 license,
    ensuring its use in accordance with
    open research and collaboration
    principles.

More information about this dataset: The
    Glue RTE dataset is derived from the
    Recognizing Textual Entailment (RTE)
    task, a subset of the GLUE benchmark.
    It is designed to evaluate models on
    the task of natural language inference,
    which involves determining if one
    sentence logically follows from
    another. In the ported version hosted
    on HuggingFace's platform, it retains
    its original purpose but introduces
    some modifications for easier
    processing. Notably, the columns
    originally named 'sentence1' and
    'sentence2' have been renamed to
    'text1' and 'text2' respectively. This
    dataset is often used to train and
    evaluate machine learning models on
    their ability to understand and process
    natural language, particularly in
    contexts where discerning relationships
    between two text statements is crucial.
    One important characteristic of the
    dataset is that its test split is
    unlabeled, with all entries in the
    label column set to -1. This is in line
    with many benchmark datasets where the
    test labels are withheld to prevent
    overfitting and encourage robust model
    evaluation.


There are other models that are evaluated
    on the dataset to judge whether the
    model and dataset are connected:
- 42dot/42dot_LLM-PLM-1.3B: 42dot LLM-PLM
    is a pre-trained language model
    developed by 42dot, designed to handle
    both Korean and English text. It is
    part of the 42dot LLM series,
    leveraging a 1.3 billion parameter
    configuration based on a Transformer
    decoder architecture, akin to LLaMA 2.
    The model is trained with a corpus of
    diverse text sources, both in Korean
    and English, aimed at providing a
    versatile foundation for various
    natural language tasks. The
    architecture consists of 24 layers, 32
    attention heads, a hidden size of 2048,
    and an FFN size of 5632. Pre-training
```

```
    involved 49,000 GPU hours using NVIDIA
    A100, with specific hyperparameters
    such as a global batch size of 4
    million tokens and a learning rate of
    4E-4. A Byte-level BPE tokenizer was
    developed from scratch to support the
    model, featuring a vocabulary size of
    approximately 50,000 tokens.
- CobraMamba/mamba-gpt-3b: The Mamba GPT-3B
    model is a fine-tuned version of the
    open-lama model, providing enhanced
    performance across several evaluation
    subtasks. It stands out as the
    best-performing 3B model, achieving
    performance metrics comparable to the
    larger llama-7b model. This model
    utilizes the transformers library and
    has been optimized for GPU usage,
    requiring the installation of specific
    versions of the transformers,
    accelerate, and torch libraries. The
    Mamba GPT-3B is designed to generate
    text with a focus on maintaining
    coherence and context, making it
    suitable for various applications that
    require advanced language processing
    capabilities.
- ModelCloud/Meta-Llama-3.1-8B-Instruct-
gptq-4bit: This model is a quantized
    version of Llama-3.1 with 8 billion
    parameters, created using the GPTQModel
    framework. It utilizes a 4-bit
    precision format, which allows for
    efficient storage and computation
    without significant loss of model
    performance. The model's configuration
    includes a group size of 128, symmetric
    quantization, and true sequential
    processing. These settings are
    optimized to maintain a balance between
    model accuracy and computational
    efficiency. The quantization process
    reduces the model's footprint, making
    it suitable for deployment in
    environments with limited resources,
    while still providing robust
    performance on various tasks. The use
    of the GPTQ quantization method ensures
    that the model can handle different
    types of data inputs efficiently.
- ibm-research/ColD-Fusion: The ColD Fusion
    model is a finetuned language model
    that builds upon the RoBERTa base
    architecture. It was trained on 35
    diverse datasets and is designed to
    serve as a robust base model for
    various natural language processing
    tasks. The model leverages a novel
    approach called ColD Fusion, which
    allows for multitask learning benefits
    through distributed computation. Unlike
    traditional multitask learning that
    requires simultaneous access to all
    datasets and substantial computational
    resources, ColD Fusion operates with
    limited communication and no data
    sharing. This enables the recycling of
    finetuned models to iteratively enhance
    the pretrained model. The resulting
    model achieves strong performance
    across the datasets it was trained on
    and serves as an excellent starting
    point for finetuning on new datasets.
    The model demonstrates a 2.45-point
    improvement over RoBERTa base without
    architectural changes and outperforms
    previous multitask models. For detailed
    insights, refer to the paper linked in
    the README.
- PygmalionAI/metharme-1.3b: Metharme 1.3B
    is an instruction-tuned language model
    based on EleutherAI's Pythia 1.4B
    Deduped. It is designed specifically
    for conversation, roleplaying, and
    storywriting with a focus on fictional
    contexts. This model can be guided
    using natural language like other
    instruct models. It has been fine-tuned
    using a mixture of regular instruction
    data, alongside roleplay and fictional
    stories, with synthetically generated
    instructions. Metharme 1.3B utilizes a
    unique prompting system with three role
    tokens: `<|system|>`, `<|user|>`, and
    `<|model|>`. The `<|system|>` token can
    provide background information,
    `<|user|>` indicates user input, and
    `<|model|>` signifies the model's
    generated response. These tokens can be
    chained to create conversation history.
    The model is suitable for entertainment
    purposes, particularly in fictional
    writing, and it is not optimized for
    safety or harmlessness. It may produce
    offensive or socially unacceptable
    content and is not reliable for factual
    accuracy.
- tensorblock/bloomz-7b1-mt-GGUF: The model
    'bloomz-7b1-mt' is a multilingual
    text-generation model based on the
    BigScience Bloom architecture. It
    supports a wide array of languages
    including common ones like English,
    Spanish, and Chinese, as well as less
    common languages such as Fon, Twi, and
    Tsonga. Additionally, it is capable of
    understanding and generating code in
    several programming languages,
    including Python, JavaScript, and C++.
    The model is designed to handle various
    natural language processing tasks such
    as coreference resolution, natural
    language inference, and sentence
    completion. It uses datasets like
    Winogrande, XWinograd, and XNLI, among
    others, to evaluate its performance
    across different tasks. The model is
    distributed under the
    bigscience-bloom-rail-1.0 license and
    is part of the 'bigscience/xP3mt'
    dataset, reflecting its broad
    multilingual and multitask focus. Its
    applications can range from sentiment
    analysis and query suggestion to the
    generation of creative content such as
    fairy tales and fables.



Please predict whether this model should be
```

```
    evaluated on this dataset. Provide your
    answer as a JSON object with two keys:
    'prediction' (a boolean, true or false)
    and 'reason' (a brief explanation of
    your reasoning).
```

Listing 3: LLM with graph attribute prediction example

```
Given a machine learning model named
    'tasksource/ModernBERT-large-nli' and a
    dataset named 'stanfordnlp/snli'.

More information about this model:
    ModernBERT is a multi-task fine-tuned
    model specifically designed for natural
    language inference (NLI) tasks. It has
    been trained on a diverse set of NLI
    datasets including MNLI, ANLI, SICK,
    WANLI, doc-nli, LingNLI, FOLIO,
    FOL-NLI, LogicNLI, and Label-NLI among
    others. The training was conducted over
    200,000 steps using an Nvidia A30 GPU,
    resulting in a powerful model for
    reasoning tasks. ModernBERT surpasses
    the performance of models like llama
    3.1 8B Instruct in specific tasks such
    as ANLI and FOLIO. It excels in long
    context reasoning, sentiment analysis,
    and zero-shot classification with new
    labels. The model is versatile,
    offering significant potential for
    fine-tuning on single tasks like SST,
    but it is particularly effective
    out-of-the-box for zero-shot
    classification and NLI tasks.

More information about this dataset: The
    Stanford Natural Language Inference
    (SNLI) corpus is a comprehensive
    dataset designed to support the task of
    natural language inference (NLI), also
    known as recognizing textual entailment
    (RTE). Version 1.0 of the dataset
    comprises 570,000 human-written
    sentence pairs labeled with entailment,
    contradiction, and neutral. These
    labels are crucial for training and
    evaluating models that aim to determine
    the inference relationship between two
    short texts. The SNLI dataset is an
    essential resource for developing
    systems that can understand and predict
    semantic relationships in natural
    language, making it a benchmark for
    text representation learning
    methodologies. The dataset was created
    through crowdsourcing efforts, with
    premises originating from the Flickr
    30k and VisualGenome corpora, and
    hypotheses generated by crowdworkers on
    Amazon Mechanical Turk. Each sentence
    pair includes a premise and a
    hypothesis, and the task is to classify
    the relationship between them. The
    dataset provides a balanced
    classification challenge, offering a
    rich source of data for training
    machine learning models in natural
    language processing applications.


tasksource/ModernBERT-large-nli's
    performance on other datasets:
- pietrolesci/nli_fever: accuracy: 0.780
    (info: The dataset in question is a
    modification of the FEVER dataset,
    tailored for Natural Language Inference
    (NLI) research. Originally, the FEVER
    dataset consisted of claims derived
    from Wikipedia, each paired with a
    label indicating the veracity of the
    claim. However, this setup did not
    align with standard NLI tasks which
    typically involve a pair of sequences
    (e.g., premise and hypothesis) mapped
    to a label. To bridge this gap and
    facilitate NLI research, the creators
    of this dataset have reformatted it to
    pair claims with textual evidence, thus
    converting it into a pair-of-sequences
    to label dataset. This transformation
    enables the application of NLI models
    on the FEVER dataset. The labels are
    mapped using predefined categories,
    where 'SUPPORTS' is mapped to
    entailment, 'NOT ENOUGH INFO' to
    neutral, and 'REFUTES' to
    contradiction. Additionally, the
    dataset includes a 'verifiable' column
    encoded to indicate whether a claim can
    be verified. Despite these
    modifications, the dataset maintains
    consistency with the original FEVER
    dataset, ensuring reliability and
    validity for research purposes.)
- yale-nlp/FOLIO: accuracy: 0.710 (info:
    The dataset is designed to provide a
    comprehensive collection of data
    relevant to the training and evaluation
    of machine learning models across
    various applications. It comprises a
    wide range of data types and
    structures, ensuring versatility and
    scalability for different research and
    development purposes. The dataset is
    curated to support both supervised and
    unsupervised learning tasks,
    facilitating experimentation with
    classification, regression, clustering,
    and more advanced machine learning
    methodologies. Each data entry in the
    dataset is meticulously labeled and
    documented, providing clear context and
    metadata to aid in the effective
    utilization of the data. The dataset is
    continually updated to incorporate the
    latest advancements and feedback from
    the user community, ensuring it remains
    relevant and useful. It is distributed
    under the MIT license, allowing for
    broad usage, adaptation, and
    distribution, making it an ideal
    resource for both academic and
    commercial projects. The dataset's
    accessibility and comprehensiveness
    make it a valuable asset for data
    scientists, researchers, and developers
    aiming to advance machine learning
    capabilities.)
```

- rediska0123/puzzte: accuracy: 0.590 (info: The dataset is designed to serve as a resource for binary question-answering models. It consists of questions that require a true or false answer. The main features of the dataset include a 'question' field, which contains the text of the question itself, and an 'answer' field, which is a boolean indicating whether the answer is true or false. The dataset is available in a single configuration named 'default' and contains a single split, 'train', which includes 911 examples. The total dataset size is 242,839 bytes, while the download size is 64,267 bytes.)
- causal-nlp/CLadder: accuracy: 0.890 (info: The dataset consists of multiple configuration files specifying different data splits. In the provided configurations, there are two main data files: 'full_v1.5_default' and 'full_v1'. These files are likely CSV formatted and are stored in the 'data' directory. The 'default' configuration suggests that it might be the primary or recommended setup for working with the dataset. Each configuration is defined under the 'configs' key, with each containing a 'config_name' and associated 'data_files'. The data files are specified with a 'split' name and the corresponding file path. This setup allows users to select different versions or subsets of the data depending on their use case, facilitating experiments with varied data distributions or updated datasets. Such configuration flexibility is crucial for testing machine learning models under different conditions or for ensuring reproducibility in research.)
- MoritzLaurer/dataset_train_nli: accuracy: 0.950 (info: The dataset comprises multiple features including 'text', 'hypothesis', 'labels', 'task_name', and 'label_text'. The 'labels' feature is a class label with two possible values: 'entailment' (0) and 'not_entailment' (1). This dataset is primarily used for natural language inference tasks, where the goal is to determine if a given hypothesis is entailed by the premise text. The data is organized into a single configuration named 'default' and focuses on a training split. The training data consists of 1,018,733 examples, occupying 315,017,473 bytes. The total download size for the dataset is approximately 206 MB. This structured setup allows for easy implementation of machine learning models aimed at understanding the entailment relationship between pairs of sentences.)
- tasksource/ruletaker: accuracy: 0.990 (info: The 'ruletaker' dataset is designed to evaluate the ability of transformer models to perform reasoning over natural language. This dataset consists of synthetic data generated to simulate logical reasoning tasks using controlled language. The primary features of the dataset include 'context', 'question', 'label', and 'config', each represented as a string, which are structured to test the model's capacity to infer and deduce truths from given premises. The dataset is provided in three splits: 'train' with 480,152 examples, 'dev' with 75,872 examples, and 'test' with 151,911 examples, collectively requiring a dataset size of 372,450,135 bytes. It is licensed under the Apache-2.0 license and primarily intended for English language processing. The dataset is based on the work presented in the paper 'Transformers as Soft Reasoners over Language' by Peter Clark, Oyvind Tafjord, and Kyle Richardson, which was presented at the Twenty-Ninth International Joint Conference on Artificial Intelligence in 2020.)
- tasksource/babi_nli: accuracy: 0.980 (info: The babi_nli dataset is designed for natural language inference tasks, specifically focusing on logical reasoning capabilities. This dataset supports various logic-based tasks by providing pairs of premises and hypotheses, each labeled as either 'entailed' or 'not-entailed.' The dataset is monolingual, using English language text, and is created through a combination of expert-generated annotations and crowdsourcing efforts. It falls under the size category with examples ranging between 1,000 and 10,000. The dataset offers multiple configurations, each targeting different aspects of logical reasoning such as agents' motivations, coreference, deduction, induction, and several others. Each configuration provides its own train, validation, and test splits, with varying dataset sizes and download sizes. The dataset can be used to evaluate models on their ability to perform natural language inference tasks, an essential component of many modern applications in natural language processing.)

stanfordnlp/snli's performance with other models:

- tasksource/ModernBERT-base-nli: accuracy: 0.830 (info: ModernBERT is a multi-task fine-tuned transformer model specifically designed for natural language inference (NLI) tasks. It incorporates datasets such as MNLI, ANLI, SICK, WANLI, doc-nli, LingNLI, FOLIO, FOL-NLI, LogicNLI, and Label-NLI, among others. The model is an 'instruct' version, optimized for tasks requiring reasoning and zero-shot

```
    classification. ModernBERT's training
    regime involved 200,000 steps on an
    Nvidia A30 GPU, enhancing its
    capabilities in long-context reasoning
    and sentiment analysis. It surpasses
    other models like llama 3.1 8B Instruct
    in specific NLI tasks such as ANLI and
    FOLIO. Though additional performance
    improvements can be achieved through
    single-task fine-tuning (e.g., SST),
    the current version excels in zero-shot
    classification and natural language
    inference across various categories
    including contradiction, entailment,
    and neutral classification. The model's
    architecture employs different
    classification heads on top of the same
    transformer, making it versatile for
    multiple applications. Notably,
    ModernBERT benefits from NLI training
    data such as the 'label-nli' dataset,
    specifically designed to enhance
    zero-shot classification abilities.)

Please predict the accuracy that
    tasksource/ModernBERT-large-nli would
    achieve on stanfordnlp/snli. Provide
    your answer as a JSON object with two
    keys: 'prediction' (a float between 0
    and 1) and 'reason' (a brief
    explanation of your reasoning).
```