KG-MFTO: Knowledge-Guided Multi-Form Optimization for Large Language Model Merging

Authors List

***Abstract*—The proliferation of large language models (LLMs) fine-tuned for specialized domains has created a pressing need for efficient methods to consolidate their capabilities. Training-free model merging has emerged as a promising solution, but existing approaches either rely on static heuristics such as Task Arithmetic, TIES, and DARE, or employ adaptive strategies like AdaMerging and Evolutionary Merge Recipes that initiate costly cold-start searches for each new merge. As the number of models and tasks grows, the search space becomes prohibitively complex, with intricate patterns of collaboration and conflict between model–model and model–task relationships. We address this challenge by formulating model merging as a Multi-Form Transfer Optimization (MFTO) problem. Rather than attempting to solve the full merge in one step, we decompose it into a curriculum of forms—simpler sub-problems defined by subsets of models, tasks, or constrained objectives. Each form yields partial solutions and exposes local synergy or interference patterns, which are transferred back to guide the global optimization. To operationalize this paradigm, we propose KG-MFTO, a framework that integrates (i) a dynamic Knowledge Graph that encodes model–model and model–task relationships, (ii) a Curriculum Planner that adaptively sequences forms to maximize information gain, and (iii) a knowledge-guided evolutionary solver that warm-starts CMA-ES with priors derived from the knowledge graph. Experiments on diverse benchmark tasks demonstrate that KG-MFTO substantially accelerates convergence and improves merge quality compared to both static merging rules and unguided evolutionary search. Ablation studies confirm the complementary contributions of the knowledge graph, curriculum planning, and dual warm-start solver. By explicitly modeling and transferring relationship knowledge across forms, KG-MFTO offers a principled and efficient path toward scalable, training-free model merging.**

***Index Terms*—Large Language Model, Model Merging, Knowledge Graph, Multiform Transfer Optimization**

I. Introduction

THE integration of specialized capabilities from multiple pre-trained large language models (LLMs) into a unified, high-performance system presents a compelling yet formidable challenge in contemporary artificial intelligence research [1]. The predominant paradigms for capability enhancement, such as full-model fine-tuning or joint training on composite datasets, are often computationally prohibitive at the scale of modern foundation models. Consequently, there is a growing imperative for efficient, training-free methodologies that can fuse the distinct competencies of existing models. This is particularly salient in scenarios where models are accessible only via black-box interfaces, precluding direct parameter manipulation. A robust training-free merging strategy would enable the modular reuse and composition of pre-trained assets across diverse domains and tasks. However, the search for an optimal amalgamation of models, each comprising billions of parameters and possessing unique, often idiosyncratic knowledge, constitutes a profoundly complex optimization problem. Rudimentary techniques such as parameter averaging or other static heuristics frequently yield suboptimal outcomes, as they fail to resolve latent parametric conflicts and preserve the full spectrum of constituent model capabilities [2].

Initial efforts to address this challenge have focused on static, training-free merging algorithms. Methodologies such as Task Arithmetic [3], which combines models by operating on their task vectors (the parameter deltas relative to a common pre-trained base), have demonstrated initial viability. Subsequent refinements, including TIES-Merging [4], introduced sign-based alignment and parameter pruning to mitigate interference between models, while DARE [5] employed dropout-inspired techniques to enhance the robustness of the merged model. Although these methods establish the feasibility of training-free fusion, they are inherently prescriptive; they define a fixed combination rule rather than optimizing the merge strategy for a specific context. As the complexity of the merging problem scales—encompassing a greater number of models, a wider array of tasks, or more heterogeneous domains—the interaction structure between model parameters becomes increasingly intricate. Certain model pairings may exhibit synergistic behavior on some tasks while creating destructive interference on others. Neglecting these fine-grained relational dynamics can lead to unstable merges characterized by catastrophic forgetting or a dilution of essential functionalities.

More advanced approaches have begun to incorporate adaptive mechanisms. AdaMerging [6], for instance, introduces trainable coefficients to dynamically weight the contribution of each model, while other works have employed evolutionary algorithms (e.g., CMA-ES) to perform a black-box search for optimal merging recipes [7]. These contributions underscore the necessity of transitioning from fixed heuristics to adaptive, search-based strategies that can account for complex model–model and model–task interactions. Nevertheless, both static and adaptive approaches are typically investigated in balanced or low-dimensional settings, where the number of candidate models is comparable to the number of evaluation tasks. This assumption simplifies the search space but diverges from real-world practice.

In realistic deployment scenarios, the number of available SFT models grows far faster than the number of well-defined tasks. Benchmarks and application workloads are relatively stable and limited, whereas new expert models are continuously produced from diverse data sources and fine-tuning protocols. The resulting ecosystem is one where #models $\gg$ #tasks: for each task, practitioners may face dozens of potential model candidates, many of which exhibit overlapping yet partially complementary strengths. This imbalance transforms the merging problem into a combinatorial explosion, where the challenge is not the scarcity of expertise but the efficient selection, weighting, and integration of many models. Existing methods, designed for smaller and more balanced regimes, are ill-suited to this large-scale setting, especially as they operate in a cold-start manner and fail to transfer knowledge across related problems.

This limitation motivates a paradigm shift towards a more principled framework capable of systematically decomposing the global merging problem, extracting relational insights from tractable sub-problems, and leveraging these insights to guide the overall optimization process. We propose to reframe the model merging problem through the lens of Multi-Form Transfer Optimization (MFTO) [8]. Within the MFTO paradigm, a complex optimization task is decomposed into a portfolio of simpler, auxiliary problems—termed "forms"—which are solved concurrently with continuous knowledge exchange. A form may represent a constrained version of the primary problem, such as merging a subset of models, optimizing for a subset of tasks, operating on a reduced-dimensional parameter space, or addressing a specific alignment sub-problem. By solving these forms, we can probe local interaction structures (e.g., synergy, conflict) in controlled environments and transfer the resulting partial solutions and relational knowledge to inform the global search.

In this work, we introduce KG-MFTO, a system that operationalizes the MFTO paradigm for training-free model merging. The cornerstone of our system is a dynamic Knowledge Graph (KG), which serves as a structured, centralized repository for intermediate solutions and inter-form relationships [9]. Nodes in the KG represent distinct forms (e.g., merging models A and B on tasks X and Y), while edges encode dependencies and relational properties. As individual forms are solved, the KG is populated with their solutions, creating a global map of the search landscape. This structured memory, in turn, informs a Curriculum Planner that strategically sequences the optimization of forms, progressing from simpler, high-yield sub-problems to more complex ones.

Each form is optimized via a guided evolutionary solver. Given the non-convex and black-box nature of the objective landscape, where gradient-based methods are inapplicable by design, we employ a population-based evolutionary search [10]. Crucially, our solver is tightly integrated with the KG. The initial population for a new form is seeded with high-quality solutions from related forms stored in the KG, thus circumventing the cold-start problem. The solver optimizes the merging configuration for the form's specific objective and contributes its best-found solution back to the KG, progressively refining the global knowledge base. This synergistic interplay between curriculum planning and knowledge-guided evolutionary search facilitates a highly efficient, gradient-free optimization process that relies solely on forward passes for evaluation.

In summary, our primary contributions are as follows:

1) A novel formulation of model merging as a Multi-Form Transfer Optimization (MFTO) problem, which decomposes the high-dimensional search into a set of tractable, interrelated sub-problems.
2) The design and implementation of KG-MFTO, a system architecture centered on a dynamic Knowledge Graph that serves as structured memory to enable explicit knowledge transfer between optimization forms.
3) A curriculum-guided evolutionary search methodology, where a Curriculum Planner adaptively sequences forms to create an optimization curriculum, and a solver leverages prior solutions from the KG to accelerate convergence.
4) Extensive empirical validation demonstrating that KG-MFTO successfully merges multiple LLMs without training, achieving performance that meets or exceeds that of individual models while significantly outperforming baseline methods in optimization efficiency.

Our work presents a new paradigm for model composition, demonstrating that complex fusion problems can be effectively addressed by decomposing them and systematically transferring knowledge between their constituent forms. We believe this approach paves the way for more general and scalable transfer optimization frameworks in machine learning.

II. Related Work

A. Model Merging and Composition without Training

Our work relates to the growing body of research on merging neural models to combine their capabilities. Early approaches to model merging include simple parameter averaging techniques. For instance, Wortsman et al. demonstrated that averaging the weights of multiple fine-tuned models (so-called Model Soup) can often improve accuracy without extra inference cost. While straightforward and sometimes effective, naive merging by linear combination provides no guarantee of retaining all task skills – in fact, arbitrary averaging can degrade performance due to destructive interference between model updates. Subsequent research has proposed more principled merging schemes. Interpolation-based methods like SLERP perform spherical linear interpolation of model weights, yielding smoother transitions between models than direct averaging. Another line of work represents model differences as task vectors, which are weight deltas obtained by subtracting a base model from a fine-tuned model. By adding or subtracting such task vectors, one can combine multiple task-specific updates in a manner analogous to arithmetic operations, enabling multitask blending or forgetting of skills. This task arithmetic approach treats merging as composing the transformations each model underwent during fine-tuning.

Beyond closed-form combinations, several optimization-based model merging methods have been introduced. These treat the merge as a search for an optimal set of merged parameters that maximize some performance metric, often subject to preserving each model's behavior on its training distribution. For example, Checkpoint Ensembling methods use techniques like Bayesian optimization to find the best weighting coefficients when averaging two models or their layers. Yadav et al. (2023) address the issue of interference between merged models' knowledge by selectively trimming and aligning weights before merging (the TIES-Merging method), effectively mitigating conflicting parameter updates. Other approaches incorporate additional data or knowledge to guide merging; for instance, some recent works perform training-free domain adaptation via merging, leveraging batch norm statistics or weight alignment to adapt a model to multiple target domains without fine-tuning. In the realm of large language models, toolkits such as MergeKit have made implementations of merging algorithms readily available, reflecting the practical demand for combining specialized LLMs. Evolutionary strategies have even been applied: an Evolutionary Model Merge technique was reported to use genetic algorithms to efficiently search the space of merging coefficients for multiple models. Overall, these works confirm that model merging can be formulated as an optimization problem – however, existing methods typically solve it in a single stage, directly attempting to find one merged model or a set of layer-wise merge ratios. In contrast, our approach introduces a multi-stage process (MFTO) where the merge is achieved via a series of sub-problems. By doing so, we avoid a brute-force search in the full parameter space and instead incrementally build a solution, which to our knowledge is a novel contribution in model composition.

B. Task Decomposition and Curriculum Optimization

Our use of multiple "forms" and a curriculum over sub-problems is inspired by ideas from task decomposition and curriculum learning. In machine learning, curriculum learning refers to training a model on a sequence of tasks or data samples ordered from easiest to hardest, which can speed up convergence and improve generalization. The classic work of Bengio et al. (2009) demonstrated that presenting tasks in a meaningful order (for example, starting with simpler concepts) leads to better learning outcomes than training on a shuffled or all-at-once curriculum. While curriculum learning is typically applied in the context of model training, the underlying principle of progressive complexity is applicable to optimization problems as well. Our Curriculum Planner extends this idea by scheduling the solution of merging sub-problems in an order that gradually increases complexity – for instance, beginning with merging smaller subsets of models or easier tasks before tackling the full merge. This can be seen as a form of problem decomposition: rather than solving the large merge in one step, we break it into a sequence of simpler merges whose results feed into the next. Such divide-and-conquer strategies have precedents in optimization. For example, the method of alternating or block coordinate optimization tackles a large problem by iteratively optimizing subsets of variables while holding others fixed; this is a common heuristic to cope with high-dimensional search spaces. Our forms can be viewed similarly – e.g., focusing on one portion of the model's parameters (or one subset of tasks) at a time – except that we do not simply freeze and cycle through variables. Instead, we solve each sub-problem to (near) optimality using our evolutionary solver, then use those partial solutions as building blocks or warm starts for larger sub-problems. There is also a connection to hierarchical and modular learning: in multi-task learning, one might train separate models or modules on different groups of tasks and then integrate them, which is analogous to our initial forms covering task subsets or model subsets. By explicitly leveraging a knowledge graph to integrate these pieces, our approach ensures the sub-solutions are combined in a coherent way, rather than leaving the integration to chance.

A concrete example of curriculum-style merging can be found in the greedy Model Soup approach. There, models are merged one by one, always merging the next best model into the current soup if it improves performance, thereby creating a sequence of intermediate merges. This resembles a manual curriculum over models. Our work generalizes this notion: rather than a fixed greedy schedule, we maintain a flexible curriculum over a rich set of forms (not just individual models, but potentially arbitrary sub-problems like "merge models A and B on task X"). The Curriculum Planner leverages information in the knowledge graph (such as which forms have yielded high performance or how similar two forms' solutions are) to decide the next merging step. This data-driven scheduling is reminiscent of automatic curriculum learning, where the sequence of tasks is decided by the learner's progress. In our case, the "learner" is the evolutionary solver and the progress is reflected in the knowledge graph. Our approach thus ties task decomposition with a feedback loop: the outcomes of earlier sub-tasks actively inform the selection of later sub-tasks.

C. Transfer Optimization and Multi-Form Methodologies

Our work is deeply rooted in the concept of transfer optimization (TO), especially as studied in evolutionary computation. TO techniques aim to accelerate the optimization of a target problem by exploiting knowledge from other related problem instances or formulations. Within this paradigm, different strategies have been explored. Sequential transfer optimization (STO) solves a source task and then transfers its solution or learned model as a starting point for a target task. Multi-task optimization (MTO) or evolutionary multitasking solves several tasks simultaneously using a shared population, allowing individuals to implicitly learn from multiple tasks at once. Multi-form transfer optimization (MFTO), by contrast, deals with alternate formulations of a single task. In MFTO, one constructs multiple versions of the problem – for example, simplified objectives or constrained variants – and either solves them in parallel or in some coordinated fashion while exchanging information. Prior work in this area often focuses on engineering design problems and multi-fidelity optimization. For instance, researchers have considered optimizing a high-fidelity objective alongside cheaper low-fidelity approximations of the same problem, using the low-fidelity solution to guide the high-fidelity search. Another example is multi-objectivization, where a single-objective problem is temporarily reformulated as a multi-objective one to help overcome local optima. In these cases, the different formulations (high vs. low fidelity, single vs. multi-objective) are the "forms" in MFTO, and knowledge (partial solutions, promising regions of the search space, etc.) is transferred among them via a shared knowledge base or by injecting solutions from one form into another. Our approach to model merging can be seen as a new application of MFTO. Each merging sub-problem we define is an alternate formulation of the ultimate merge task, differing either in the subset of models involved, the subset of tasks evaluated, or the constraints imposed (e.g. limiting which layers are merged). We maintain continuous knowledge transfer between these forms through the Knowledge Graph, which plays the role of the central knowledge base facilitating transfer. This explicit tracking of cross-form knowledge distinguishes our method from earlier evolutionary MFTO implementations, which sometimes relied on implicit transfer (such as using a common population for all forms). By using a graph-structured memory, we can represent complex relationships – for example, that a solution to form A (merging models 1 and 2 on task X) is relevant to form B (merging models 1, 2, and 3 on task X) or form C (merging models 1 and 2 on a broader task Y). Our work thus contributes to the transfer optimization literature by demonstrating the efficacy of form-level decomposition in a high-dimensional, discrete optimization scenario (neural weight space), and by introducing mechanisms to guide the transfer effectively.

It is important to contrast our approach with meta-learning. Meta-learning, in a broad sense, also leverages experience from past learning episodes (tasks) to improve learning on new tasks. However, meta-learning frameworks (e.g. MAML) typically assume a distribution of tasks and aim to learn a meta-model or initialization that can quickly adapt to a previously unseen task. In our case, we do not aim to generalize to arbitrary new tasks; instead we address one fixed target problem – merging a specific set of models for specific tasks – and we leverage forms of that same problem to solve it efficiently. There is no separate "meta-training" and "meta-testing" phase; all our sub-problems ultimately serve the single goal of solving the target merge. This means our setting is not meta-learning at all, but rather a single-task optimization enriched by transfer from auxiliary formulations. This perspective aligns more closely with transfer optimization and curriculum problem solving than with learning-to-learn paradigms. By removing the meta-learning framing, we highlight that KG-MFTO is solving a one-off optimization problem (albeit with internal structure), not learning a general strategy for future problems. This clarity is important for positioning our contribution: the innovation lies in how we decompose and transfer within a complex problem, not in any meta-learner that spans tasks.

D. Evolutionary Search with Knowledge Guidance

Finally, our methodology connects with prior work on using evolutionary algorithms (EAs) and other heuristics for optimizing neural networks and other high-dimensional systems. Evolutionary and genetic algorithms have a long history in optimizing neural network weights and architectures ("neuro-evolution"), particularly when gradient-based methods are infeasible or to avoid getting trapped in poor local minima. In recent years, EAs have been applied to problems like neural architecture search, hyperparameter tuning, and even direct weight merging. For example, Real et al. (2019) showed that evolution strategies can effectively discover convolutional network architectures competitive with those found by gradient-based NAS, illustrating the power of guided random search in discrete design spaces. In model merging specifically, as mentioned earlier, one approach applied a genetic algorithm to find optimal merge ratios for combining multiple language models. Our use of an evolutionary solver is in the same spirit, but with two key differences: guidance and granularity. First, we guide the evolutionary search using domain-specific knowledge stored in the KG. This relates to the idea of memetic algorithms or knowledge-informed EAs, where problem-specific heuristics or learned knowledge are injected into the evolutionary process to improve efficiency. Surrogate-assisted EAs, for instance, build regression models (surrogates) to estimate fitness and focus search on promising regions; by analogy, our knowledge graph can be seen as a learned model of the search space, indicating promising configurations or combinations based on past evaluations. We explicitly initialize populations and design crossover/mutation strategies in light of known good partial solutions from the KG, making our evolutionary search non-blind – it exploits structure uncovered during the search itself. Second, rather than running one global EA on the full problem, we run many smaller EAs on the forms. This is reminiscent of cooperative co-evolution strategies, where sub-populations evolve different parts of a solution and co-adapt. In cooperative co-evolution, periodically the partial solutions are combined to evaluate the whole solution's fitness. KG-MFTO's approach can be seen as a dynamic form of cooperative co-evolution: each form's evolutionary run optimizes a part of the problem (with other parts implicitly fixed or simplified), and the knowledge graph ensures that these parts ultimately come together consistently. Notably, some recent works in evolutionary multi-tasking have started to use graph-based structures to analyze or direct knowledge transfer between tasks. We build on this idea by constructing an actual graph-based planner for the optimization process. The result is a highly distributed yet coordinated search procedure: multiple evolutionary searches tackle different forms, but through the knowledge graph they share information and converge towards a common merged model solution.

In summary, our approach is uniquely situated at the intersection of model merging, transfer/curriculum optimization, and evolutionary search. We advance the state of the art by treating model merging as an MFTO problem and by introducing a tangible knowledge-driven mechanism (the KG and curriculum planner) to implement form-level knowledge transfer. In the following section, we detail the KG-MFTO methodology, describing how forms are generated and linked via the knowledge graph, how the curriculum planner operates, and how the evolutionary solver is configured for guided merging. We also distinguish how these components function from a transfer optimization standpoint, rather than any meta-learning interpretation.

III. Method

We frame training-free model merging as an online meta-optimization problem. Instead of solving each merge instance in isolation, our framework—*Knowledge Graph-centric Multi-form Transfer Optimization* (KG-MFTO)—accumulates structured experience and transfers it to subsequent instances. KG-MFTO follows a repeated *perceive–plan–act–learn* loop: (i) synchronize a dynamic knowledge graph (KG), (ii) select an informative sub-problem (*form*) via a curriculum planner, (iii) solve the form with a knowledge-guided evolutionary optimizer, and (iv) update the KG and train a graph neural network (GNN) online.

A. Problem Setup and Merge Abstraction

Let $\mathcal{M} = \{M_1, \dots, M_n\}$ be SFT experts and $\mathcal{T} = \{\tau_1, \dots, \tau_k\}$ evaluation tasks with validation splits. A training-free merge operator $\mathrm{Merge}(\cdot, \boldsymbol{\alpha})$ combines model parameters using coefficients $\boldsymbol{\alpha} \in \mathbb{R}^n$, e.g., parameter-space arithmetic such as TIES-Merging [?] or DARE [?]. We define an evaluation functional $\mathcal{F}$ that aggregates task scores:

$$\begin{aligned}\boldsymbol{\alpha}^\star &\in \arg\max_{\boldsymbol{\alpha}\in C} \mathcal{F}\big(\mathrm{Merge}(\mathcal{M}, \boldsymbol{\alpha}), \mathcal{T}\big) \\ &= \arg\max_{\boldsymbol{\alpha}\in C} \sum_{j=1}^{k} w_j \,\mathrm{Score}\big(\mathrm{Merge}(\mathcal{M}, \boldsymbol{\alpha}), \tau_j\big),\end{aligned} \tag{1}$$

where $w_j \geq 0$, $\sum_j w_j = 1$, and C encodes coefficient constraints induced by the merge operator (e.g., simplex Δ^{n-1} for convex combinations; signed weights if delta-arithmetic is permissible).

a) Forms and curriculum.: Directly optimizing (1) is expensive as n and k grow. KG-MFTO decomposes optimization into *forms*, $f = (\mathcal{M}_f, \mathcal{T}_f)$ with $\mathcal{M}_f \subseteq \mathcal{M}$ and $\mathcal{T}_f \subseteq \mathcal{T}$, organized by a curriculum level ℓ that typically increases $|\mathcal{M}_f|$ (e.g., pairs $\rightarrow$ triples $\rightarrow$ larger subsets). Solving simpler forms amortizes knowledge useful for larger ones.

B. Dynamic Heterogeneous Knowledge Graph

The KG is a heterogeneous graph $G = (V, E)$ with node types $\{\textsc{Model}, \textsc{Task}\}$ and edge types $\{\textsc{Model–Model}, \textsc{Model–Task}\}$.

a) Nodes.: Each model node M_i has a learnable embedding $\mathbf{h}_i^M \in \mathbb{R}^d$ (randomly initialized). Each task node τ_j has a fixed semantic embedding $\mathbf{h}_j^T \in \mathbb{R}^{d_T}$ obtained with an encoder such as SimCSE [?].

b) Edges and sufficient statistics.: A $\textsc{Model–Model}$ edge $e(M_i, M_{i'})$ stores (i) an exponential-moving-average (EMA) synergy score $s_{ii'} \in \mathbb{R}$ and (ii) an uncertainty $u_{ii'} \in \mathbb{R}_{\geq 0}$. A $\textsc{Model–Task}$ edge $e(M_i, \tau_j)$ stores a performance statistic π_{ij} (e.g., EMA of M_i's score on τ_j) and optional uncertainty v_{ij}.

Given an evaluated form f with realized score $y_f = \mathcal{F}(\mathrm{Merge}(\mathcal{M}_f, \boldsymbol{\alpha}_f), \mathcal{T}_f)$ and recipe $\boldsymbol{\alpha}_f$, we update $\textsc{Model–Model}$ synergies by

$$\begin{aligned} g_f(i,i') &= \big(y_f - \bar{y}_f^{\mathrm{ref}}\big) \cdot \psi(\alpha_{f,i}, \alpha_{f,i'}), \\ s_{ii'} &\leftarrow (1-\lambda) s_{ii'} + \lambda\, g_f(i,i'), \end{aligned} \tag{2}$$

where $\bar{y}_f^{\mathrm{ref}}$ is a reference (e.g., best single-expert or uniform-merge score on $\mathcal{T}_f$), and $\psi(a,b)$ is a bounded co-contribution weight, e.g., $\psi(a,b) = 2ab$ for simplex weights (larger when both experts contribute). Edge uncertainty decays with evidence, e.g.,

$$\begin{aligned} n_{ii'} &\leftarrow n_{ii'} + \mathbb{I}\{i,i' \in \mathcal{M}_f\}, \\ u_{ii'} &\leftarrow \frac{\kappa}{\kappa + n_{ii'}} \quad \text{or} \quad u_{ii'} \leftarrow \rho u_{ii'} + (1-\rho)\,\hat{\sigma}_{ii'}^2, \end{aligned} \tag{3}$$

where $n_{ii'}$ counts observations and $\hat{\sigma}_{ii'}^2$ is an empirical variance proxy across forms containing (i, i').

Algorithm 1: Candidate generation at level ℓ

```
Input  : Experts $\mathcal{M}$, tasks $\mathcal{T}$, KG $G$, level $\ell$, budgeted pool size $K$
Output : Candidate set $C_\ell$
1  $C_\ell \leftarrow \emptyset$
2  while $|C_\ell| < K$ do
3      if Bernoulli($p_{unc}$) then
4          Propose $f$ s.t. $\mathcal{M}_f$ contains at least one pair with top-$q$ uncertainty in $G$
5      else if Bernoulli($p_{syn}$) then
6          Propose $f$ by growing a near-clique under current $\hat{S}$ around a seed model
7      else
8          Propose $f$ uniformly at random among $|\mathcal{M}_f| = \ell$ subsets
9      end if
10     Ensure $|\mathcal{M}_f| = \ell$ and minimal task subset $\mathcal{T}_f$ covers target metrics
11     $C_\ell \leftarrow C_\ell \cup \{f\}$
12 end while
13 return $C_\ell$
```

C. Graph Neural Network for Relational Reasoning

We use a heterogeneous GAT [?] with type-specific projections: $\tilde{\mathbf{h}}_i^M = W_M \mathbf{h}_i^M$, $\tilde{\mathbf{h}}_j^T = W_T \mathbf{h}_j^T$, followed by L edge-aware attention layers. Edge features (e.g., $s_{ii'}$, $u_{ii'}$, π_{ij}) modulate attention logits. For a candidate form $f = (\mathcal{M}_f, \mathcal{T}_f)$, we pool its induced subgraph into three heads:

1) **Performance head:** predicts $\hat{y}_f$ as a surrogate to $\mathcal{F}$.
2) **Alpha head:** produces $\hat{\boldsymbol{\alpha}}_f$ over $\mathcal{M}_f$. We implement this as a small Transformer that attends over the model-node embeddings of $\mathcal{M}_f$ to handle variable cardinality.
3) **Synergy head:** outputs a symmetric $\hat{S}_f \in \mathbb{R}^{m\times m}$ with $m = |\mathcal{M}_f|$, where $(\hat{S}_f)_{ii'}$ estimates pairwise interaction within the form.

The online loss combines these objectives:

$$\mathcal{L}_{\mathrm{GNN}} = w_{\mathrm{perf}}(\hat{y}_f - y_f)^2 + w_\alpha\, \mathrm{KL}\big(\boldsymbol{\alpha}_f^\star \,\|\, \hat{\boldsymbol{\alpha}}_f\big) + w_{\mathrm{syn}} \big\|\hat{S}_f - S_f^{\mathrm{KG}}\big\|_F^2, \tag{4}$$

where $\boldsymbol{\alpha}_f^\star$ is the best recipe observed for f to date, and S_f^{KG} is a target synergy matrix derived from the current KG (e.g., $(S_f^{\mathrm{KG}})_{ii'} \leftarrow s_{ii'}$ for $i, i' \in \mathcal{M}_f$). After each evaluation, we perform a few gradient steps on (4).

D. Curriculum Planner: Acquisition over Forms

At iteration b, the planner scores a candidate set C_ℓ at curriculum level ℓ with a UCB-style acquisition:

$$\begin{aligned} A(f) &= \hat{y}_f + \beta_b\, \sigma(f), \\ \sigma(f) &= \mathrm{Agg}\big(\{u_{ii'} : i, i' \in \mathcal{M}_f\}\big), \end{aligned} \tag{5}$$

where Agg is a robust aggregator (mean or trimmed mean), and β_b anneals (e.g., $\beta_b = \beta_0/\sqrt{1+b}$). Candidates are generated by a mixed strategy: (i) uncertainty targeting (include at least one high-uncertainty pair), (ii) synergy densification (near-cliques under $\hat{S}_f$), and (iii) curriculum expansion (advance ℓ when pairwise coverage for the next level exceeds a threshold). We pick $f^\star = \arg\max_{f\in C_\ell} A(f)$.

Algorithm 2: Knowledge-guided solver for a single form

```
Input  : Form $f$, GNN outputs $(\hat{y}_f, \hat{\alpha}_f, \hat{S}_f)$, operator
         Merge, evaluator $\mathcal{F}$
Output : Best recipe $\alpha^\star_f$ and score $y_f$
1 $(\mathbf{m}_0, C_0) \leftarrow$ WarmStart$(\hat{\alpha}_f, \hat{S}_f)$          // Eqs. (6)
2 $\alpha^\star_f \leftarrow$ CMAES$\big(\mathbf{m}_0, C_0,\ \mathbf{z} \mapsto$
    EvaluateMerge(ProjToCoeffs$(\mathbf{z})$, Merge, $\mathcal{F}$)$\big)$
3 $y_f \leftarrow$ EvaluateMerge$(\alpha^\star_f$, Merge, $\mathcal{F})$
4 return $(\alpha^\star_f, y_f)$
```

E. Knowledge-Guided Solver: Dual Warm-Start CMA-ES

For the selected form $f = (\mathcal{M}_f, \mathcal{T}_f)$, we maximize $\mathcal{F}(\mathrm{Merge}(\mathcal{M}_f, \alpha), \mathcal{T}_f)$ over $\alpha \in \mathcal{C}_f$ using CMA-ES [?] with a dual warm-start derived from GNN predictions.

a) Parameterization and constraints.: We optimize an unconstrained vector $\mathbf{z} \in \mathbb{R}^m$ ($m = |\mathcal{M}_f|$) and map to coefficients via (i) simplex reparameterization $\alpha = \mathrm{softmax}(\mathbf{z})$ if $\mathcal{C}_f = \Delta^{m-1}$, or (ii) signed map if deltas allow negatives, e.g., $\alpha = \tanh(\mathbf{z})/\sum_i |\tanh(z_i)|$.

b) Mean warm-start.: Set the initial mean to $\mathbf{m}_0 = \mathcal{R}(\hat{\alpha}_f)$, where $\mathcal{R}$ is the inverse map from coefficients to $\mathbf{z}$ (e.g., logit for softmax with small ε-clipping for stability).

c) Covariance warm-start.: Map $\hat{S}_f$ to a correlation matrix R:

$$R_{ii'} = \begin{cases} 1, & i = i', \\ \phi\big((\hat{S}_f)_{ii'}\big), & i \neq i', \end{cases} \qquad \phi(x) = \tanh(\gamma x) \in (-1, 1). \tag{6}$$

Enforce positive semidefiniteness via eigenvalue clipping: $R = Q\Lambda Q^\top$, $\Lambda \leftarrow \max(\Lambda, \epsilon I)$ with small $\epsilon > 0$. Scale with initial step size σ_0: $C_0 = \sigma_0^2 R$. The initial sampling distribution is $\mathcal{N}(\mathbf{m}_0, C_0)$.

d) Noisy evaluation and restarts.: We use median-of-means over mini-batches of validation items to reduce evaluator noise. If CMA-ES stagnates (no improvement for G_{stall} generations), we trigger a trust-region restart with reduced σ_0; if warm-start appears misleading (rapid covariance blow-up or repeated rejections), we fall back to diagonal $C_0 = \sigma_0^2 I$.

F. Closed-Loop Learning and Graph Updates

After solving f, we (i) update MODEL–MODEL synergies and uncertainties via (2)–(3), (ii) refresh MODEL–TASK statistics π_{ij} using task-wise scores of the merged model, and (iii) train the GNN with loss (4). We then decide whether to advance the curriculum level ℓ based on coverage criteria, e.g., a minimum number of observations per pair in the next level's induced cliques.

G. End-to-End Procedure

The full loop is summarized in Algorithm 4. We track a global budget B of form evaluations, and at each iteration generate candidates (Alg. 1), score them with (5), solve the best one using Alg. 2, update the KG (Alg. 3), and adapt the curriculum.

Algorithm 3: Graph update after evaluating a form

```
Input  : KG $G$, form $f = (\mathcal{M}_f, \mathcal{T}_f)$, recipe $\alpha^\star_f$, score $y_f$,
         reference $\bar{y}^{\text{ref}}_f$
Output : Updated KG $G$
1 for each $(i, i') \subset \mathcal{M}_f$ do
2     $g \leftarrow (y_f - \bar{y}^{\text{ref}}_f) \cdot 2\alpha^\star_{f,i}\alpha^\star_{f,i'}$
3     $s_{ii'} \leftarrow (1-\lambda)s_{ii'} + \lambda g$;  $n_{ii'} \leftarrow n_{ii'} + 1$;  $u_{ii'} \leftarrow \kappa/(\kappa + n_{ii'})$
4 end
5 for each $i \in \mathcal{M}_f,\ j \in \mathcal{T}_f$ do
6     Update $\pi_{ij}$ with the task-wise score; optionally update $\upsilon_{ij}$
7 end
8 GNN_TrainStep $(G, \{(f, \alpha^\star_f, y_f)\})$
9 return $G$
```

Algorithm 4: KG-MFTO: knowledge-guided multi-form transfer optimization

```
Input  : Experts $\mathcal{M}$, tasks $\mathcal{T}$, merge operator Merge, budget
         $B$
Output : Set of solved forms and recipes $\{(f, \alpha^\star_f, y_f)\}$
1  Initialize KG $G$ with nodes for $\mathcal{M}$ and $\mathcal{T}$; initialize GNN;
     $\ell \leftarrow 2$; $b \leftarrow 0$
2  while $b < B$ do
3      $\mathcal{C}_\ell \leftarrow$ GenerateCandidates$(\mathcal{M}, \mathcal{T}, G, \ell, K)$
4      for $f \in \mathcal{C}_\ell$ do
5          $(\hat{y}_f, \hat{\alpha}_f, \hat{S}_f) \leftarrow$ GNN_Predict$(G, f)$
6          $\sigma(f) \leftarrow \mathrm{Agg}(\{u_{ii'} : i, i' \in \mathcal{M}_f\})$
7          $A(f) \leftarrow \hat{y}_f + \beta_b\, \sigma(f)$
8      end for
9      $f^\star \leftarrow \arg\max_{f \in \mathcal{C}_\ell} A(f)$
10     $(\alpha^\star_{f^\star}, y_{f^\star}) \leftarrow$ SolveForm$(f^\star, \hat{\alpha}_{f^\star}, \hat{S}_{f^\star}$, Merge, $\mathcal{F})$
11     $G \leftarrow$ UpdateGraph$(G, f^\star, \alpha^\star_{f^\star}, y_{f^\star}, \bar{y}^{\text{ref}}_{f^\star})$
12     $b \leftarrow b + 1$;  if ReadyToAdvance $(G, \ell)$ then
13         $\ell \leftarrow \ell + 1$
14     end if
15 end while
```

H. Practical Extensions and Stability

a) Layer-/block-wise coefficients.: When the merge operator admits layer-wise or block-wise weights, we use a low-rank parameterization to control dimensionality: $\alpha_i^{(\ell)} \propto \mathrm{softmax}(a_i + b_\ell)$, with per-model a_i and per-layer b_ℓ, optionally with a small interaction term. CMA-ES then optimizes $\{a_i\}$ and $\{b_\ell\}$ instead of a full $n \times L$ tensor.

b) Task weighting.: If tasks differ in scale or reliability, we use uncertainty-aware weights $w_j \propto 1/\widehat{\mathrm{Var}}(\mathrm{Score}(\cdot, \tau_j))$ with normalization, or apply rank-based normalization before aggregation in $\mathcal{F}$.

c) Cold-start and calibration.: At initialization ($b = 0$), we set $s_{ii'} = 0$, $u_{ii'} = 1$, and calibrate heads by self-supervision on individual models: $\hat{y}$ regresses to π_{ij} marginals, alpha head defaults to uniform, and synergy head to zero off-diagonal.

d) Robustness.: We monitor covariance condition number in CMA-ES; if it exceeds a threshold, we shrink C by $C \leftarrow (1-\eta)C + \eta\,\sigma^2 I$. For numerical stability, logits for the softmax parameterization are clipped to $[-\zeta, \zeta]$ before mapping.

I. Complexity and Amortization

For a form with $m = |\mathcal{M}_f|$, GNN inference is $O(|E_f|d)$ with sparse attention; candidate scoring is $O(|\mathcal{C}_\ell|)$ once predictions are cached. CMA-ES per generation costs $O(\lambda m^2)$ for covariance updates with λ offsprings; wall-clock is dominated by λ evaluations of $\mathcal{F} \circ \text{Merge}$. The curriculum and warm-start reduce both the number of generations and required candidate pool sizes, amortizing cost across forms as the KG matures.