PersonalPlan: Planning Multi-Agent Systems for Personalized Programming Learning

Anonymous ACL submission

Abstract

Effective programming education requires personalized instruction adapted to diverse learner backgrounds. However, while LLM-based multi-agent systems (MAS) excel at complex planning, existing planners often lack profile-grounding and pedagogical scaffolding, thereby undermining personalized programming learning. To fill in the gap, we first introduce **MAP-PPL** (**M**ulti-**A**gent **P**lans for **P**ersonalized **P**rogramming **L**earning), a profile-conditioned multi-agent planning datasetwith 3,043 query–profile instances from 1,730 Stack Overflow question groups and 2,738 learner profiles. Each plan specifies agents, subtasks, executable steps, and prerequisite dependencies. Then, we propose **PersonalPlan**, a two-stage MAS planner that first performs hierarchical SFT with separate LoRA adapters for profile-aware task decomposition and step dependency planning, then applies a trajectory-level GRPO to encourage model generating executable, personalized, and pedagogically scaffolded plans. Extensive experiments on MAP-PPL comparing PersonalPlan against frontier LLMs, generic MAS frameworks, and agentic planners demonstrate its superiority. With only 8B and 32B variants, PersonalPlan achieves state-of-the-art plan executability, personalization, and pedagogical quality, effectively orchestrating MAS for agent-student interactions.

1 Introduction

Programming literacy is increasingly useful beyond traditional computer science classrooms. With AI assistants, learners from diverse backgrounds ask models to automate analyses, inspect code, connect APIs, debug workflows, or turn informal goals into executable programs (Denny et al., 2024; Hsu, 2025). Programming learning often requires resource retrieval, prerequisite explanation, code demonstration, execution, testing, debugging, and reflection (Barua et al., 2014; Ponzanelli et al., 2013). This motivates a planning view of programming learning, in which a learner query should be mapped to coordinated instructional roles, subtasks, tools, and dependencies.

Recent code tutors and LLM-based educational systems instantiate parts of this process through guardrailed coding help, Socratic debugging, or multi-role tutoring (Kazemitabaar et al., 2024; Liffiton et al., 2024; Kargupta et al., 2024; Zhao et al., 2025; David and Ghosh, 2026). However, they rarely produce the specialized plans required by personalized programming MAS, which must satisfy three core requirements. First, plans should be *personalized in structure*: learners with different prior knowledge and professional contexts may need different agent roles, subtask granularity, and prerequisite paths (Nabizadeh et al., 2021; Chen, 2008; Jiang et al., 2022). Second, they must be *executable*: fluent surface plans can still contain cyclic dependencies, missing prerequisite links, invalid tool calls, or agent–tool mismatches in execution(Kim et al., 2024; Wei, 2025; Xiong et al., 2025). Third, they should encode *pedagogical scaffolding* as checkable structure, making concept-before-application ordering, code demonstration, validation, and feedback checkpoints explicit before the tutoring workflow runs (Hmelo-Silver et al., 2007; Sentance et al., 2019).

In this paper, we propose **PersonalPlan**, a profile-conditioned multi-agent planning framework for personalized programming-learning MAS. A full tutoring plan couples two decisions: the high-level instructional scaffold and the low-level executable workflow. PersonalPlan therefore uses hierarchical SFT with two LoRA adapters: *Profile-Aware Decomposition* (PAD) selects learner-conditioned subtasks and pedagogical agents, while *Step Dependency Planning* (SDP) grounds this scaffold into step actions, prerequisite edges, agent–step assignments, and tool bindings. A lightweight joint-alignment stage then exposes

SDP to PAD-produced scaffolds and gives PAD a downstream-compatible scaffold signal, reducing the hierarchical exposure-bias mismatch between gold scaffolds used during SFT and generated scaffolds seen at inference. Finally, the reward-adative GRPO optimizes complete plans with verifiable composable rewards for structural validity, profile grounding, and pedagogical phase coverage, while a hard gate penalizes schema violations, cycles, and invalid tool bindings.

To facilitate PersonalPlan and broader research on profile-conditioned MAS planning, we also construct **MAP-PPL** (**M**ulti-**A**gent **P**lans for **P**ersonalized **P**rogramming **L**earning), a dataset of 3,043 query–profile–plan instances from 1,730 Stack Overflow question groups and 2,738 learner profiles across seven learning intents (Beyer et al., 2018). Each instance pairs a learner profile and correctness reference with a strict JSON MAS plan generated by an advanced LLM (Claude Sonnet 4.6 (Anthropic, 2026)) that specifies agents, subtasks, executable steps, tool bindings, execution order, and dependency edges. All released plans pass deterministic schema and DAG checks plus an execution-effectiveness gate, and the corpus records profile-grounded structure and pedagogical scaffolding for evaluating whether planners adapt agent roles, subtask decomposition, dependencies, and teaching phases to learner background.

We evaluate PersonalPlan on MAP-PPL against frontier LLM, generic MAS, and agentic workflow planners on both static plan-quality metrics and dynamic MAS execution with tutor–learner interactions. With only 8B and 32B variants, PersonalPlan achieves state-of-the-art plan executability, personalization, and pedagogical quality, effectively orchestrating MAS for agent-student interactions. Ablations show that the two-stage SFT and reward-guided refinement are both critical: single-stage SFT with only decomposition or dependency supervision fails to produce executable plans, while joint alignment and GRPO each contribute to improving tool binding and pedagogical quality. To summarize, our contributions are threefold:

1. We introduce **PersonalPlan**, a profile-conditioned multi-agent planning framework for personalized programming-learning MAS that combines hierarchical SFT and reward-adative GRPO with verifiable rewards for executable structure, profile grounding, and pedagogy.

2. We construct **MAP-PPL**, a dataset of 3,043 query–profile–plan instances from 1,730 Stack Overflow question groups and 2,738 learner profiles across seven learning intents to facilitate related research. All plans are executable and contain profile-grounded structure and pedagogical scaffolding for evaluating personalized programming-learning MAS planners.

3. Experiments show that PersonalPlan improves the three target properties of personalized programming-learning MAS: executability better tool-bound plans and compact executed traces; personalization through stronger profile fit and profile-sensitive variation; and pedagogy through better instructional quality and realized tutoring behavior[1].

2 Related Work

LLM-based multi-agent planning. LLM-based multi-agent systems decompose tasks across specialized agents that communicate, call tools, and coordinate through conversations or workflow graphs. General frameworks such as AutoGen, CAMEL, AgentVerse, MetaGPT, ChatDev, and AutoAgents instantiate agent teams for broad task solving (Wu et al., 2023; Li et al., 2023; Chen et al., 2024b; Hong et al., 2024; Qian et al., 2024; Chen et al., 2024a), while AIPOM, AFlow, AOP, and WorFBench generate explicit agent/task graphs (Kim et al., 2025; Zhang et al., 2025a; Li et al., 2025; Qiao et al., 2025). Planner-training methods further improve high-level plans with trajectories, hierarchical supervision, and post-training rewards (Yin et al., 2024; Zhao et al., 2024; Xiong et al., 2025; Parmar et al., 2025). These works motivate our MAS and workflow-planning baselines, but they mainly optimize generic completion or orchestration; learner profiles rarely determine the agent roster, prerequisite graph, tool bindings, or instructional phases.

Personalized programming education. Personalized tutoring adapts hints, resources, problem sequences, or learning paths to learner knowledge and goals (Graesser et al., 2004; Piech et al., 2015; Nabizadeh et al., 2021; Chen, 2008). Recent LLM educational agents add lesson planning, learner simulation, Socratic teaching, and pedagogy-aware RL (Zhang et al., 2025b; Liu et al., 2024; Peng

[1]Our code and data will be released upon acceptance.

et al., 2025; Dinucu-Jianu et al., 2025). For programming education, CodeAid, CodeHelp, TreeInstruct, AdaCoder, CodeEdu, and IntelliCode provide guardrailed coding help, Socratic debugging, and multi-role tutoring (Kazemitabaar et al., 2024; Liffiton et al., 2024; Kargupta et al., 2024; Zhu et al., 2025; Zhao et al., 2025; David and Ghosh, 2026). However, their adaptation is usually a dialogue policy, hint choice, recommendation, or session outline rather than an inspectable MAS execution plan, making it difficult to validate whether agent roles, tool bindings, dependencies, and teaching phases form an executable personalized tutoring workflow. Therefore, PersonalPlan fills this gap by generating learner-conditioned MAS plans optimized for executability, profile grounding, and pedagogy.

3 PersonalPlan

3.1 Problem Formulation

Let I_q denote a query raised by a learner and I_p denote the learner profile containing skills, knowledge, prior experience, and background context. The goal is to generate a structured multi-agent plan $\mathcal{P} = (\mathcal{A}, \mathcal{T}, \mathcal{S}, G)$ for a tutoring-oriented MAS that executes a personalized instructional workflow, where $\mathcal{A} = \{a_1, \ldots, a_N\}$ is a set of specialized agents, $\mathcal{T} = \{\tau_1, \ldots, \tau_M\}$ is a set of subtasks, and $\mathcal{S} = \{s_1, \ldots, s_K\}$ denotes the executable steps. Additionally, the prerequisite relations among steps define a dependency graph $G = (\mathcal{S}, \mathcal{E})$, where $\mathcal{E} = \{(s_i, s_j) : s_i \in \mathrm{dep}(s_j)\}$. Our objective is to learn a planner $\mathcal{F}_\theta : (I_q, I_p) \mapsto \mathcal{P}$ under three requirements: executable MAS structure with schema-valid references and an acyclic G; personalization to I_p in agents, subtasks, and instructions; and pedagogical scaffolding (Hmelo-Silver et al., 2007) through concept-before-application ordering (Sentance et al., 2019) and explicit solution checks.

To solve this problem, PersonalPlan uses a two-stage training pipeline: hierarchical profile-aware SFT first learns the profile-conditioned scaffold and step-dependency structure, and GRPO then refines full-plan generation with verifiable rewards for structural validity, learner grounding, and pedagogical coverage, as shown in Figure 1.

3.2 Hierarchical SFT

An MAS plan orchestrate tutoring MAS for personalized programming learning combines two decisions: constructing the instructional scaffold, including agents and subtasks, and grounding it into executable steps, dependencies, and tool use. Because one-stage generation entangles these signals and fixes execution details before the pedagogy is stable, we factorize the planning into two LoRA (Hu et al., 2022) stages: Profile-Aware Decomposition for scaffold generation and Step Dependency Planning for executable workflow construction.

Profile-Aware Decomposition (PAD). PAD predicts the high-level scaffold $(\mathcal{T}, \mathcal{A})$ from the query–profile pair (I_q, I_p). For each training instance, the input is $x_{\text{pad}} = I_q \oplus I_p$, and the target sequence is $y_{\text{pad}} = \mathrm{ser}(\mathcal{T}^\star, \mathcal{A}^\star)$, where $\mathrm{ser}(\cdot)$ serializes the gold subtasks $\mathcal{T}^\star$ and pedagogical agents $\mathcal{A}^\star$ into the target text format. The PAD supervision signal is constructed from the high-level components of the gold plan only, excluding step-level workflow details. PAD is trained by minimizing the sequence-level autoregressive SFT objective:

$$\mathcal{L}_{\text{pad}} = -\sum_{(x_{\text{pad}}, y_{\text{pad}}) \in \mathcal{D}_{\text{pad}}} \log P_{W_{\text{pad}}}\left(y_{\text{pad}} \mid x_{\text{pad}}\right). \quad (1)$$

Here, $W_{\text{pad}} = W_0 + \Delta W_{\text{pad}}$, where W_0 denotes the frozen backbone parameters and ΔW_{pad} is the trainable low-rank LoRA update.

Step Dependency Planning (SDP). SDP grounds the high-level scaffold into an executable workflow by predicting the step set and dependency graph $G = (\mathcal{S}, \mathcal{E})$ from I_q, I_p, and the gold scaffold $(\mathcal{T}^\star, \mathcal{A}^\star)$. For each training instance, the input is $x_{\text{sdp}} = I_q \oplus I_p \oplus \mathcal{T}^\star \oplus \mathcal{A}^\star$, and the target sequence is $y_{\text{sdp}} = \mathrm{ser}(\mathcal{S}^\star, \mathcal{E}^\star)$. Thus, the target contains only step-level gold components, while the gold scaffold is used as context to isolate SDP from early PAD errors and learn dependencies, agent–step assignments, tool bindings, and execution order under a clean scaffold. SDP is trained by minimizing the sequence-level autoregressive SFT objective:

$$\mathcal{L}_{\text{sdp}} = -\sum_{(x_{\text{sdp}}, y_{\text{sdp}}) \in \mathcal{D}_{\text{sdp}}} \log P_{W_{\text{sdp}}}\left(y_{\text{sdp}} \mid x_{\text{sdp}}\right). \quad (2)$$

Here, $W_{\text{sdp}} = W_0 + \Delta W_{\text{sdp}}$, where ΔW_{sdp} is the trainable low-rank LoRA update for step-dependency planning.

Joint Alignment. Although PAD and SDP are trained independently, directly composing them creates a distribution mismatch: SDP is trained on the gold scaffold $z^\star = (\mathcal{T}^\star, \mathcal{A}^\star)$ but receives

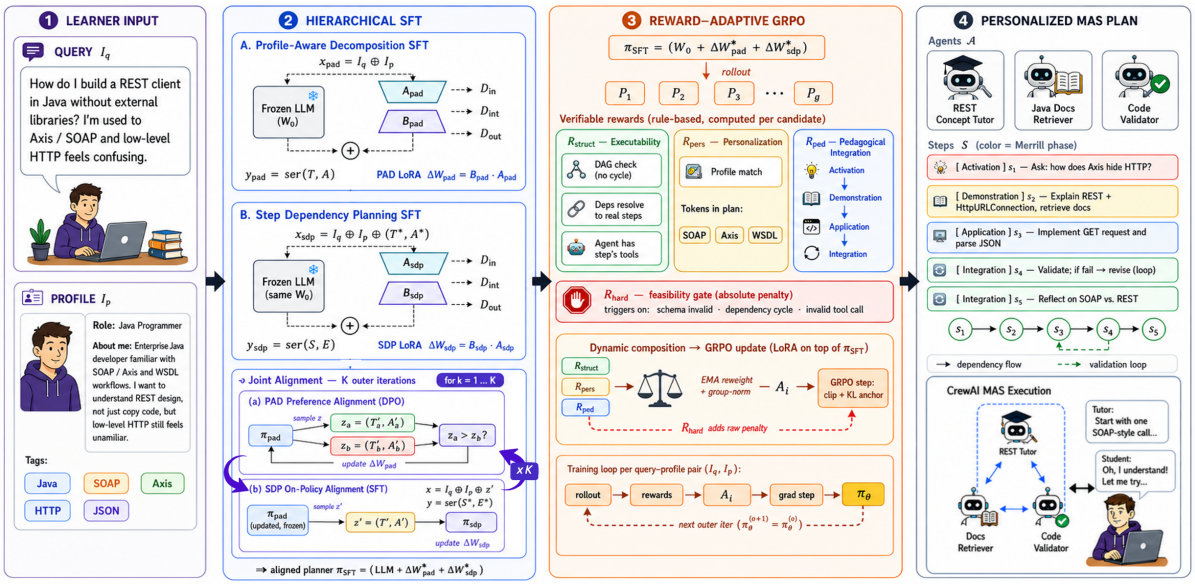


Figure 1: Overview of PersonalPlan.

the PAD-generated scaffold $z' = (\mathcal{T}', \mathcal{A}')$ at inference. This hierarchical exposure bias (Bengio et al., 2015; Ranzato et al., 2016) leaves SDP with unseen contexts and gives PAD no downstream grounding signal. We therefore add a lightweight joint alignment stage following on-policy hierarchical alignment and distillation work (Yin et al., 2024; Erdogan et al., 2025; Agarwal et al., 2024): over K batched outer iterations, PAD is preference-aligned toward gold-like scaffolds, and SDP is then adapted to PAD-produced scaffolds. Appendix A.1.1 gives the candidate scoring rules and losses.

3.3 Reward-Adaptive GRPO

Hierarchical SFT provides token-level imitation, but plan-level properties such as acyclicity, valid bindings, profile grounding, and pedagogical coverage must be optimized over complete trajectories. We therefore use *Reward-Adaptive GRPO* (Shao et al., 2024): for each (I_q, I_p), the model samples plans $\{\mathcal{P}_i\}_{i=1}^{g}$ and scores each parsed plan with three verifiable soft rewards plus a hard feasibility gate. Rewards depend only on $\mathcal{P}_i$ and I_p—not a gold plan or LLM judge—and soft rewards are normalized within the rollout group while non-recoverable feasibility failures are handled separately. We next introduce the reward components and GRPO training procedure[2].

Structural Reward. An MAS plan $\mathcal{P}_i$ is structurally executable only if its steps admit an acyclic execution order, all declared step prerequisites resolve to valid steps, and each step is assigned to an agent whose tools and capabilities can support its execution. To make executability a verifiable training signal, we combine three rule-based predicates into an equally weighted score:

$$R_i^{\text{struct}} = \tfrac{1}{3}\big(\text{DAG}(\mathcal{P}_i) + \text{DC}(\mathcal{P}_i) + \text{ATR}(\mathcal{P}_i)\big). \quad (3)$$

$\text{DAG}(\mathcal{P}_i) \in \{0, 1\}$ indicates whether the induced dependency graph is acyclic, $\text{DC}(\mathcal{P}_i) \in [0, 1]$ is the fraction of declared prerequisite edges that resolve to existing steps, and $\text{ATR}(\mathcal{P}_i) \in [0, 1]$ combines a step-level agent–tool validity check with a plan-level capability-coverage check.

Personalization Reward. For personalized programming tutoring, an MAS plan should reflect concrete learner-profile signals $\mathcal{C}(I_p)$ (*e.g.*, keywords from self-description that indicate skills, knowledge, prior experience, and background context) in its agent roster, subtask design, and step execution. To reward this profile grounding, we define the personalization reward as the fraction of profile signals covered by the plan content:

$$R_i^{\text{pers}} = \frac{|\{c \in \mathcal{C}(I_p) : c \text{ appears in } \text{Text}(\mathcal{P}_i)\}|}{\max(1, |\mathcal{C}(I_p)|)}, \quad (4)$$

where $\text{Text}(\mathcal{P}_i)$ denotes the concatenated agent, subtask, and step text of the generated plan (*e.g.*, agent name, subtask instruction, expected output).

Pedagogy Reward. To make the MAS plans function as tutoring strategies, we reward coverage

[2] Appendix A.1.2 gives the exact predicates, token sets, and phase detectors used for all reward subcomponents.

of a complete instructional cycle: probing prior knowledge, demonstrating the relevant concept or API, guiding application/implementation, and checking the solution. These stages align with Merrill's first principles of instruction (Merrill, 2002): ACTIVATION, DEMONSTRATION, APPLICATION, and INTEGRATION. Let $\mathcal{M}$ denote this phase set, and let $\mathrm{phase}(\mathcal{P}_i) \subseteq \mathcal{M}$ denote the subset detected in the generated plan $\mathcal{P}_i$. The pedagogy reward is the fraction of required phases covered by the plan:

$$R_i^{\mathrm{ped}} = \frac{|\,\mathrm{phase}(\mathcal{P}_i)|}{|\mathcal{M}|}. \tag{5}$$

In particular, to encourage interactive plans, we require $\text{ACTIVATION} \in \mathrm{phase}(\mathcal{P}_i)$ only when an ACTIVATION-matched step also requires explicit learner input.

Hard Feasibility Penalty. Although the soft rewards encourage structurally valid, personalized, and pedagogically complete plans, they are preference signals rather than feasibility constraints. Following constraint-style multi-objective RL (Huang et al., 2025), we apply a hard penalty to plans that would cause the downstream MAS to crash before execution because of schema parsing errors, dependency cycles, or invalid tool assignments:

$$R_i^{\mathrm{hard}} = -\lambda_{\mathrm{hard}}\, \mathbb{I}\big[\mathrm{SchemaInvalid}(\mathcal{P}_i) \vee \mathrm{HasCycle}(G_i) \vee \mathrm{InvalidToolCall}(\mathcal{P}_i)\big], \tag{6}$$

where G_i is the dependency graph induced by $\mathcal{P}_i$.

Dynamic Composition. Optimizing structure, personalization, and pedagogy jointly requires combining rewards with different scales and learning speeds. Following dynamic multi-objective reward balancing (Huang et al., 2025), we design a two-step composition. First, for each soft reward axis, we convert raw rewards into relative z-scores within the rollout group for the same query–profile pair. Concretely, for each $k \in \{\mathrm{struct}, \mathrm{pers}, \mathrm{ped}\}$,

$$\bar{R}_i^k = \frac{R_i^k - \mu_g(R^k)}{\sigma_g(R^k) + \varepsilon}, \tag{7}$$

where $\mu_g(R^k)$ and $\sigma_g(R^k)$ are the mean and standard deviation over candidates in that rollout group, and ε is a small numerical stabilizer. Second, we maintain normalized soft-reward weights w_k over the three axes. During training, an exponential moving average (EMA) tracks each axis's recent raw reward, and a multiplicative update increases w_k for axes that lag behind the target level. The composed per-rollout reward is

$$R_i = \sum_{k \in \{\mathrm{struct},\mathrm{pers},\mathrm{ped}\}} w_k \bar{R}_i^k + R_i^{\mathrm{hard}}. \tag{8}$$

Only the soft rewards are normalized and dynamically reweighted; R_i^{hard} remains an absolute penalty, so invalid plans cannot compensate with high personalization or pedagogy scores.

GRPO Training. When training with GRPO on the composed rewards, we increase the likelihood of higher-reward plans while keeping the updated policy close to the SFT reference model. For each input $I = (I_q, I_p)$, GRPO converts the composed rewards within its rollout group into group-relative advantages $A_i = (R_i - \mu_R)/(\sigma_R + \varepsilon)$, where μ_R and σ_R are computed over the g sampled plans for the same I. Let o_i denote the token sequence of the i-th sampled plan and $o_{i,t}$ its t-th token. The token-level probability ratio is

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid I, o_{i,<t})}{\pi_{\theta_{\mathrm{old}}}(o_{i,t} \mid I, o_{i,<t})}, \tag{9}$$

where $\pi_{\theta_{\mathrm{old}}}$ is the policy before the current update. We then optimize

$$\mathcal{L}_{\mathrm{GRPO}}(\theta) = -\mathbb{E}_{I,\{o_i\}}\Big[\tfrac{1}{g}\sum_i \tfrac{1}{|o_i|}\sum_t \min\big(r_{i,t}(\theta)A_i, \mathrm{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)A_i\big)\Big] + \beta\, \mathbb{D}_{\mathrm{KL}}[\pi_\theta \| \pi_{\mathrm{SFT}}]. \tag{10}$$

The clipping parameter ϵ limits the size of each policy update, and the fixed KL term weighted by β keeps the reward-optimized policy close to the model behavior learned during Hierarchical SFT.

4 MAP-PPL

4.1 Dataset Overview

To support profile-conditioned MAS planning for programming education, we construct **MAP-PPL** (**M**ulti-**A**gent **P**lans for **P**ersonalized **P**rogramming **L**earning). It contains 3,043 query–profile–plan instances $\{I_q, I_p, \mathcal{P}\}$ from 1,730 Stack Overflow question groups and 2,738 unique learner profiles. Each plan $\mathcal{P}{=}(\mathcal{A}, \mathcal{T}, \mathcal{S}, G)$ specifies agents, subtasks, executable steps, and a dependency graph. Stack Overflow duplicate-question groups are useful because they create one-to-many cases: the technical problem can remain fixed while the learner profile changes (Barua et al., 2014; Beyer et al., 2018).

Statistic	Value
Query–profile–plan instances	3,043
Unique question groups	1,730
Unique learner profiles	2,738
Questions with ≥ 2 profiles	971 (56.1%)
Same-question profile pairs	1,777
Agents per plan (mean / max)	2.91 / 4
Steps per plan (mean / max)	10.38 / 23
Executable / DAG-valid plans	3,043 (100.0%)
Inter-agent dependency edges	58.5%
Merrill all-phase coverage	2,974 / 3,043 (97.7%)

Table 1: Headline statistics for **MAP-PPL**. Extended distributions and diagnostics are in Appendix A.2.

Data Collection and Plan Generation. We trace duplicate links in the `original_questions` field, retain cleaned questions with accepted or strongly upvoted answers, and filter out one-line lookups, weak answers, and profiles without concrete personalization signals. This yields 3,062 query–profile pairs for generation and 3,043 final plans after validation. Claude Sonnet 4.6 receives the question, profile, and answer as correctness reference, infers the learner's starting point, and emits strict JSON with `agents`, `subtasks`, `steps`, `execution_order`, and explicit `depends_on` edges. We retain only plans that pass deterministic schema checks and an execution-effectiveness gate. Input distributions, prompts, and validation audits are reported in Appendix A.2.3, Appendix A.2.2, and Appendix A.2.9.

4.2 Dataset Characteristics

Personalization. MAP-PPL supports controlled profile-counterfactual analysis. Figure 2 shows that concrete profile skills appear in 39.1% of agent-role names, 55.2% of subtask descriptions, and 90.1% of step-level instructions. Holding the programming question fixed and swapping only the learner profile produces substantial structural rewrites: 68.0% agent-role divergence, 77.1% subtask divergence, and 71.0% step-text divergence across 1,777 same-question profile pairs.

Pedagogy and executability. Plans align with Merrill's first principles (Merrill, 2002): they activate prior knowledge, demonstrate the target concept or API, guide application, and integrate the result through validation or transfer. Tutor, retriever/-docs, and validator/checker families cover 98.1% of agent declarations and co-occur in 2,534 plans (83.3%). At the graph level, all released plans are

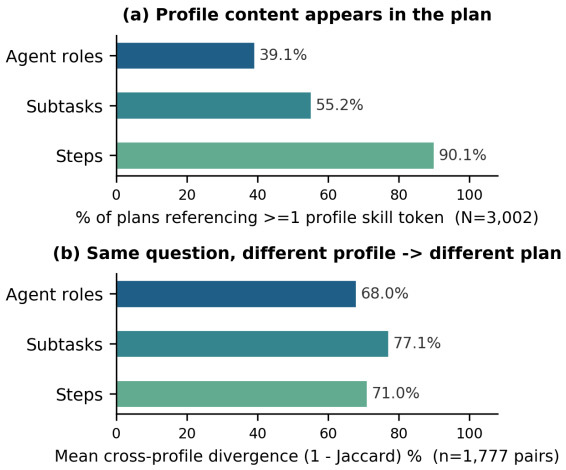


Figure 2: Profile grounding and profile-conditioning effects in **MAP-PPL**.

schema-valid, DAG-valid, and executable under the MAS executor, with non-trivial prerequisite depth, schedulable parallelism, and cross-agent coordination. Detailed order and executability diagnostics are in Appendix A.2.8 and Appendix A.2.4.

5 Experiments

To validate whether PersonalPlan can generate executable and personalized multi-agent tutoring plans, we design experiments that compare it with state-of-the-art MAS planning methods.

5.1 Baselines and Implementation

We compare PersonalPlan with three groups of planner baselines: frontier LLM planners (GPT-5.4 (OpenAI, 2025), Claude Opus 4.6 (Anthropic, 2026), and Qwen3-Max (Yang et al., 2025)), generic MAS framework planners (AutoGen (Wu et al., 2023) and AutoAgents (Chen et al., 2024a)), and agentic workflow planners (AIPOM (Kim et al., 2025) and AFlow (Zhang et al., 2025a)). These groups cover direct LLM planning, general-purpose agent orchestration, and explicit workflow construction; the selection rationale and excluded tutoring systems are detailed in Appendix A.3.

For fair comparison, all open or framework-based baselines and PersonalPlan use Qwen3-32B-Instruct as the backbone, share the same tool pool, and are converted to the same MAP-PPL plan schema. Detailed implementation settings, the shared baseline plan-generation prompt, and the evaluation metrics are provided in Appendix A.3 and Appendix A.4.

5.2 Data Splits

We split MAP-PPL by `question_id`, so all learner profiles attached to the same programming question stay in the same partition. This prevents profile variants of one question from leaking across training and testing. The split is stratified by the number of profiles per question, preserving the one-to-many structure that is central to profile-conditioned planning. The held-out test set contains 305 query–profile–plan instances from 173 question groups, and all final comparisons are reported on this split. The remaining 2,738 instances are used to train PersonalPlan.

5.3 Evaluation Method

We evaluate PersonalPlan and all baselines in two categories. *Static Plan Quality* measures properties that can be inspected before running a real MAS. It covers three aspects: *(1) Executability*, which checks schema admission through retry cost (**Atps**), residual repair rate after three reruns (**RR**), tool binding quality (**TBQ**), and dependency-graph similarity to the gold plan (**TS**); *(2) Personalization*, which measures LLM-judged profile fit (**Pers.**) and two profile-counterfactual probes: profile-induced structural variation (**PVS**) and intended-profile advantage over swapped profiles (**PNG**); and *(3) Pedagogy*, which combines prerequisite-respecting progression, no-direct-answer guidance, Merrill-phase coverage, and profile-appropriate instructional style into a composite **Ped.** score.

Plan Execution Quality measures outcomes after the generated plan is executed by an MAS and produces an agents–learner trace. For all methods, we instantiate the same CrewAI runtime: GPT-4o agents execute the planned roles and interact with GPT-4o-mini as a simulated learner conditioned on the learner profile. We report deterministic structural compactness (**SCS**), dialogue-level pedagogical quality (**PQS**), and post-dialog solve rate based on final-code correctness (r_{sol}). In addition, we report profile-conditioned learner satisfaction (**Sati.**) as an LLM-judge-based metric. All metrics are defined in Appendix A.4, and all judge-based metrics are supported by rubrics and anti-hacking safeguards in Appendix A.6.

6 Result Analysis

6.1 Static Plan Quality

Table 2 shows that **PersonalPlan** has the strongest static profile before execution. The 32B variant is best on TBQ (0.74), Pers. (0.76), PVS (0.57), and Ped. (0.65), while remaining fully admissible (Atps 1.00, RR 0.00); the 8B variant is best on TS (0.82) and tied-best on PNG (0.24). Frontier LLMs match isolated metrics, but not the full profile. Generic MAS and workflow planners preserve some structure yet trail on Pers. (0.26–0.31), PNG (0.02–0.08), and Ped. (0.31–0.35), indicating that personalized tutoring plans need explicit profile and pedagogy training signals.

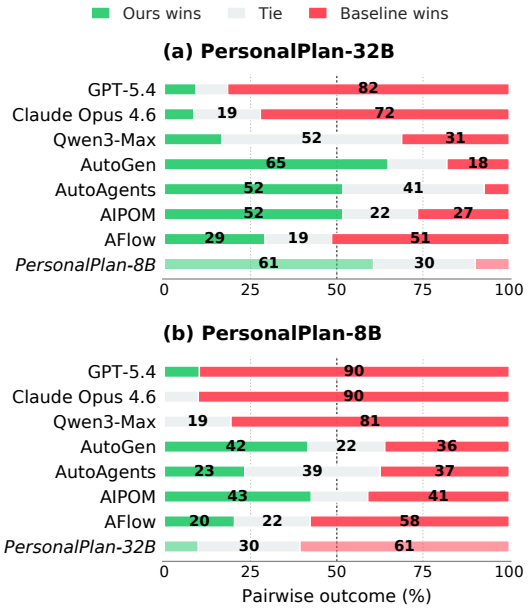


Figure 3: **Pairwise preferences over executed interaction traces.** GPT-5.4, Claude Opus 4.6, and Gemini 3.1 Pro judge, over the 305 MAP-PPL instances, whether the PersonalPlan trace or each baseline trace better supports the learner after execution. Bars report the percentage of wins, ties, and baseline wins.

6.2 Plan Execution Results

The post-execution block of Table 2 tests whether static plans are really executable for an tutoring MAS, using SCS for compactness, PQS for realized pedagogy, and r_{sol} for final-code correctness.

PersonalPlan separates these dimensions. The 8B model has the highest SCS (0.88), while the 32B model gives the strongest balanced execution profile with the best PQS (0.56) and tied-best solve rate (0.92). Figure 3 agrees: PersonalPlan-32B is preferred over most non-frontier baselines, including AutoGen (65% wins) and AutoAgents/AIPOM (52% wins), and beats the 8B variant in 61% of comparisons. It remains competitive with Qwen3-Max through many ties (52%), but still trails GPT-

Table 2: Evaluation results of PersonalPlan and baselines. Arrows indicate whether lower (↓) or higher (↑) values are better.

Method	**Static Plan Quality**								**Plan Execution Results**		
	Executable structure				**Personalization**			**Pedagogy**			
	Atps↓	RR↓	TBQ↑	TS↑	Pers.↑	PVS↑	PNG↑	Ped.↑	SCS↑	PQS↑	r_{sol} ↑
Frontier LLM planners											
GPT-5.4 (OpenAI, 2025)	1.00	0.00	0.55	0.43	0.53	0.47	0.10	0.62	0.35	0.33	0.90
Claude Opus 4.6 (Anthropic, 2026)	1.09	0.04	0.64	0.59	0.67	0.39	0.24	0.55	0.26	0.39	0.81
Qwen3-Max (Yang et al., 2025)	1.06	0.00	0.61	0.75	0.49	0.29	0.22	0.47	0.46	0.20	0.81
Generic MAS framework planners											
AutoGen (Wu et al., 2023)	1.04	0.01	0.61	0.76	0.29	0.37	0.08	0.31	0.84	0.38	0.83
AutoAgents (Chen et al., 2024a)	2.12	0.03	0.63	0.71	0.28	0.34	0.03	0.33	0.75	0.35	0.92
Agentic workflow planners											
AIPOM (Kim et al., 2025)	2.03	0.01	0.58	0.79	0.31	0.33	0.02	0.35	0.55	0.40	0.81
AFlow (Zhang et al., 2025a)	2.06	0.01	0.56	0.79	0.26	0.40	0.06	0.33	0.36	0.42	0.89
PersonalPlan (Qwen3-8b-Instruct)	1.03	0.00	0.69	**0.82**	0.70	0.48	**0.24**	0.61	**0.88**	0.36	0.77
PersonalPlan (Qwen3-32b-Instruct)	**1.00**	**0.00**	**0.74**	0.81	**0.76**	**0.57**	0.22	**0.65**	0.82	**0.56**	**0.92**

5.4 and Claude Opus 4.6. Overall, PersonalPlan yields executable learner interactions with stronger realized pedagogy than generic MAS or workflow planners, while the strongest frontier LLM planners remain difficult baselines.

6.3 Ablation Study

The ablation analysis focuses on *static plan quality*: executable structure, personalization, and pedagogy. This isolates the training pipeline itself, where PAD/SDP build the skeleton, joint alignment stabilizes it, and GRPO refines tool binding and topology.

Figure 4 shows that GRPO supplies the main final gains. For 8B, TBQ rises from 0.55 after joint alignment to 0.69, TS from 0.78 to 0.82, and Ped. from 0.54 to 0.61; for 32B, the same step improves TBQ from 0.69 to 0.74, TS from 0.77 to 0.81, and Ped. from 0.55 to 0.65. PAD-SFT and SDP-SFT provide partial structure, and joint alignment improves the skeleton, but reward optimization is needed for the final tool-use, topology, and pedagogy gains. Scale is complementary: 32B is strongest on TBQ, Pers., and Ped., while 8B remains slightly ahead on TS.

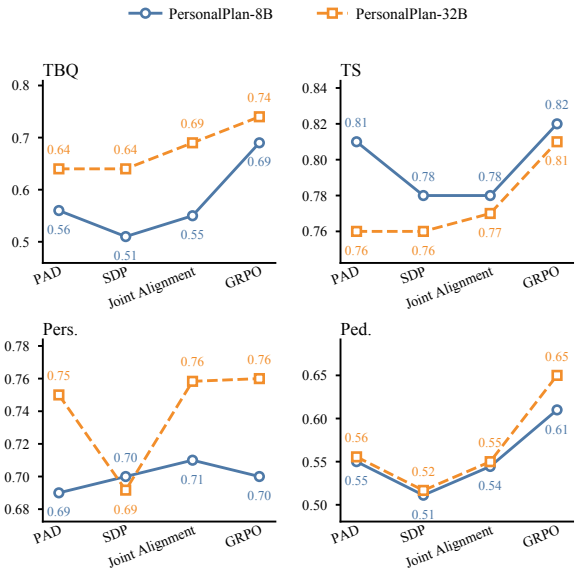


Figure 4: **Metric-wise ablation trends.** Static-quality scores for 8B and 32B variants across PAD-SFT, SDP-SFT, joint alignment, and GRPO. Pers. and Ped. are normalized to the same $[0, 1]$ scale as the structural metrics for visualization.

7 Conclusion

PersonalPlan studies personalized programming support as an inspectable multi-agent planning problem rather than a single-turn tutoring response. We introduce MAP-PPL, a profile-conditioned benchmark whose plans specify agents, subtasks, executable steps, and prerequisite graphs, and propose a two-stage planner that combines profile-aware decomposition, step dependency planning, joint alignment, and reward-adaptive GRPO. Across static plan metrics, MAS execution, and profile-conditioned preference evaluation, PersonalPlan improves executable structure, profile grounding, and pedagogical scaffolding over generic MAS and workflow-planning baselines, showing that learner-aware supervision and verifiable plan-level rewards are effective ingredients for personalized programming education.

Limitations

Currently, the system generates and executes plans without live input from the learner. Although our online simulation offers a controlled proxy, it cannot fully capture the unpredictability of real student behavior. Moving forward, the most logical extension is to ground the system in real-world interactions, such as treating per-step solve-rate gains as verifiable rewards to continuously align the pedagogy (Dinucu-Jianu et al., 2025). This would also allow us to test the system's adaptability to learner feedback and evolving needs, which is crucial for personalized education.

Ethical Considerations

MAP-PPL is constructed from public Stack Overflow content and profile text. Before release, profile fields should be filtered for personally identifying details, normalized to coarse skill and background attributes, and distributed under licenses compatible with the source platform. PersonalPlan is designed as a planning assistant for educators or learners, not as an automated replacement for instruction; generated plans can encode incorrect assumptions about a learner's prior knowledge, so deployments should keep teacher or learner review in the loop. Because profile-conditioned generation may amplify stereotypes if profiles contain sensitive attributes, the released benchmark and model card should document filtering rules, supported profile fields, and intended educational use.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. 2024. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR)*.

Anthropic. 2026. Claude Opus 4.6 model card. https://www.anthropic.com/claude.

Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical software engineering*, 19(3):619–654.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28.

Stefanie Beyer, Christian Macho, Martin Pinzger, and Massimiliano Di Penta. 2018. Automatically classifying posts into question categories on Stack Overflow. In *Proceedings of the 26th Conference on Program Comprehension*, pages 211–221.

Chih-Ming Chen. 2008. Intelligent web-based learning system with personalized learning path guidance. *Computers & Education*, 51(2):787–814.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Borje F Karlsson, Jie Fu, and Yemin Shi. 2024a. Autoagents: A framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen-Ming Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, and 1 others. 2024b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Jones David and Shreya Ghosh. 2026. IntelliCode: A multi-agent LLM tutoring system with centralized learner modeling. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 129–138. Association for Computational Linguistics.

Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing education in the era of generative ai. *Communications of the ACM*, 67(2):56–67.

David Dinucu-Jianu, Jakub Macina, Nico Daheim, Ido Hakimi, Iryna Gurevych, and Mrinmaya Sachan. 2025. From problem-solving to teaching problem-solving: Aligning LLMs with pedagogy using reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Kwon, Pawel Wawrzynski, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. ArXiv:2503.09572.

Arthur C Graesser, Shulan Lu, George Tanner Jackson, Heather Hite Mitchell, Mathew Ventura, Andrew Olney, and Max M Louwerse. 2004. Autotutor: A tutor with dialogue in natural language. *Behavior Research Methods, Instruments, & Computers*, 36(2):180–192.

Cindy E. Hmelo-Silver, Ravit Golan Duncan, and Clark A. Chinn. 2007. Scaffolding and achievement in problem-based and inquiry learning: A response

to Kirschner, Sweller, and Clark (2006). *Educational Psychologist*, 42(2):99–107.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Hsiao-Ping Hsu. 2025. From programming to prompting: Developing computational thinking through large language model-based generative artificial intelligence. *TechTrends*, 69:485–506.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Anonymous Huang and 1 others. 2025. MO-GRPO: Mitigating reward imbalance in multi-objective GRPO. *arXiv preprint arXiv:2509.22047*.

Bing Jiang, Xinyi Li, Shengyingjie Yang, Yiyao Kong, Wenlong Cheng, Chenchen Hao, and Qiyun Lin. 2022. Data-driven personalized learning path planning based on cognitive diagnostic assessments in MOOCs. In *Applied Sciences*.

Priyanka Kargupta, Ishika Agarwal, Dilek Hakkani-Tur, and Jiawei Han. 2024. Instruct, not assist: LLM-based multi-turn planning and hierarchical questioning for Socratic code debugging. In *Findings of the Association for Computational Linguistics: EMNLP 2024*.

Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara J Ericson, David Weintrop, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*.

Hannah Kim, Kushan Mitra, Chen Shen, Dan Zhang, and Estevam Hruschka. 2025. AIPOM: Agent-aware interactive planning for multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An LLM compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2024. RewardBench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.12712*.

Zheng Li, Bowen Tian, Junnan Yang, Yating Lu, Pengfei Zhou, and Yi Chen. 2025. AOP: Agent-oriented planning for decomposable tasks. *arXiv preprint arXiv:2502.09003*.

Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2024. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*.

Binwen Liu, Jiexi Ge, and Jiamin Wang. 2025. Vaiage: A multi-agent solution to personalized travel planning. *arXiv preprint arXiv:2505.10922*.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhengyuan Liu, Stella Xin Yin, Geyu Lin, and Nancy F. Chen. 2024. Personality-aware student simulation for conversational intelligent tutoring systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

M. David Merrill. 2002. First principles of instruction. *Educational Technology Research and Development*, 50(3):43–59.

Amir Hossein Nabizadeh, José Paulo Leal, Hamed N. Rafsanjani, and Rajiv R. Shah. 2021. Learning path personalization and recommendation methods: A survey of the state-of-the-art. *Expert Systems with Applications*, 159:113596.

OpenAI. 2025. GPT-5 system card. `https://openai.com/index/gpt-5-system-card/`.

Mihir Parmar, Palash Goyal, Xin Liu, Yiwen Song, Mingyang Ling, Chitta Baral, Hamid Palangi, and Tomas Pfister. 2025. PLAN-TUNING: Post-training language models to learn step-by-step planning for complex problem solving. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Xian Peng, Pan Yuan, Dong Li, Junlong Cheng, Qin Fang, and Zhi Liu. 2025. KELE: A multi-agent framework for structured Socratic teaching with large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J. Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. In *NeurIPS*.

Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack overflow in the ide. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1295–1298. IEEE.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2025. Benchmarking agentic workflow generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*.

Sue Sentance, Jane Waite, and Maria Kallia. 2019. Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education*, 29(2–3):108–135.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yifei Wang, Cha Ji, Mengmeng Wang, Yiding Liu, and Yuhong Wang. 2025. GenMentor: Tutoring with personalized learning goals and agentic multi-module workflows. In *Proceedings of the Web Conference (WWW)*.

Anonymous Wei. 2025. Beyond ReAct: A planner-centric framework for complex tool-augmented LLM reasoning. *arXiv preprint arXiv:2505.00001*.

Qifan Wu, Mohit Bansal, and 1 others. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Weimin Xiong, Yifan Song, Qingxiu Dong, Bingchan Zhao, Feifan Song, Xun Wang, and Sujian Li. 2025. MPO: Boosting LLM agents with meta plan optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

An Yang and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent Lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*. ArXiv:2311.05657.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2025a. AFlow: Automating agentic workflow generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Xueqiao Zhang, Chao Zhang, Jianwen Sun, Jun Xiao, Yi Yang, and Yawei Luo. 2025b. Eduplanner: Llm-based multi-agent systems for customized and intelligent instructional design. *IEEE Transactions on Learning Technologies*.

Jianing Zhao, Peng Gao, Jiannong Cao, Zhiyuan Wen, Chen Chen, Jianing Yin, Ruosong Yang, and Bo Yuan. 2025. CodeEdu: A multi-agent collaborative platform for personalized coding education. *arXiv preprint arXiv:2507.13814*.

Qi Zhao, Haotian Fu, Chen Sun, and George Konidaris. 2024. EPO: Hierarchical LLM agents with environment preference optimization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*.

Yueheng Zhu, Chao Liu, Xuan He, Xiaoxue Ren, Zhongxin Liu, Ruwei Pan, and Hongyu Zhang. 2025. Adacoder: An adaptive planning and multi-agent framework for function-level code generation. *arXiv preprint arXiv:2504.04220*.

A Appendix

A.1 Method Details

This appendix first records the method-side details that complement Section 3: the joint-alignment losses and the exact reward computation used by Reward-Adaptive GRPO. These details are placed before the dataset and experiment appendices so that the appendix order follows the main text.

A.1.1 Joint Alignment Details

This appendix expands the lightweight joint alignment stage introduced in Section 3. Each outer iteration alternates between a PAD preference update and an SDP on-policy update.

PAD Preference Alignment. Let π_{pad} denote the distribution induced by the current PAD adapter over $z = (\mathcal{T}, \mathcal{A})$. For each query–profile pair, we sample two candidates $z_a = (\mathcal{T}'_a, \mathcal{A}'_a)$ and

$z_b = (\mathcal{T}'_b, \mathcal{A}'_b)$ from $\pi_{\text{pad}}(\cdot \mid I_q, I_p)$. Each candidate is scored against the gold $z^\star$ with a rule-based structural similarity:

$$s(z) = \tfrac{1}{2}\,\text{Jaccard}(\mathcal{T}', \mathcal{T}^\star) + \tfrac{1}{2}\,\text{Jaccard}(\mathcal{A}', \mathcal{A}^\star), \quad (11)$$

where both subtasks and agent rosters are compared as sets. The higher-scoring candidate is treated as the preferred output z^+ and the lower-scoring one as the rejected output z^-. We update the PAD adapter ΔW_{pad} by minimizing

$$\begin{aligned} r_{\text{pad}}(z) &= \log \pi_{\text{pad}}(z \mid I_q, I_p) - \log \pi^{\text{ref}}_{\text{pad}}(z \mid I_q, I_p), \\ \mathcal{L}^{\text{ja}}_{\text{pad}} &= -\mathbb{E}_{(z^+, z^-)}[\log \sigma(\beta(r_{\text{pad}}(z^+) - r_{\text{pad}}(z^-)))]. \end{aligned} \quad (12)$$

Here, $\pi^{\text{ref}}_{\text{pad}}$ is the reference distribution before alignment, and β is the temperature parameter.

SDP On-Policy Alignment. After PAD Preference Alignment, we freeze the updated PAD adapter ΔW_{pad} and adapt SDP to PAD-produced scaffolds. For each query–profile pair, we sample $z' = (\mathcal{T}', \mathcal{A}') \sim \pi_{\text{pad}}(\cdot \mid I_q, I_p)$ and construct $x^{\text{ja}}_{\text{sdp}} = I_q \oplus I_p \oplus z'$, while the target remains the gold step-dependency serialization $y^\star_{\text{sdp}} = \text{ser}(\mathcal{S}^\star, \mathcal{E}^\star)$. We fine-tune the existing SDP LoRA ΔW_{sdp} by minimizing

$$\mathcal{L}^{\text{ja}}_{\text{sdp}} = -\mathbb{E}_{z' \sim \pi_{\text{pad}}(\cdot \mid I_q, I_p)} \log P_{W_{\text{sdp}}}\left(y^\star_{\text{sdp}} \mid I_q, I_p, z'\right). \quad (13)$$

This update adapts SDP to the PAD distribution observed at inference time.

A.1.2 GRPO Reward Computation Details

This appendix specifies the four reward components and the composition rule introduced in Section 3.3. All rewards are deterministic functions of the candidate plan and learner profile; none reads a reference plan or calls a learned model at training time.

Structural Reward. Let $\mathcal{P}_i$ denote a candidate plan and $G_i = (\mathcal{S}_i, \mathcal{E}_i)$ its step-dependency graph induced from `depends_on` fields. Following Eq. 3, the structural reward is

$$R^{\text{struct}}_i = \tfrac{1}{3}\big(\text{DAG}(\mathcal{P}_i) + \text{DC}(\mathcal{P}_i) + \text{ATR}(\mathcal{P}_i)\big).$$

This appendix specifies each of the three plan-level predicates.

Acyclicity (DAG). A linear-time cycle detector based on topological-sort traversal reports whether G_i contains any directed cycle:

$$\text{DAG}(\mathcal{P}_i) = \mathbb{I}[G_i \text{ admits a topological order}] \in \{0, 1\}. \quad (14)$$

Dependency Completeness (DC). A declared edge $(u, v) \in \mathcal{E}_i$ is valid when its source resolves to a known step in $\mathcal{S}_i$:

$$\text{DC}(\mathcal{P}_i) = \begin{cases} 1 & \text{if } \mathcal{E}_i = \emptyset, \\ \dfrac{|\{(u,v) \in \mathcal{E}_i : u \in \mathcal{S}_i\}|}{|\mathcal{E}_i|} & \text{otherwise.} \end{cases} \quad (15)$$

Agent–Tool Relevance (ATR). ATR averages a local step-level validity check (subscore A) and a plan-level capability coverage check (subscore B):

$$\text{ATR}(\mathcal{P}_i) = \tfrac{1}{2}\big(\text{ATR}_{\text{A}}(\mathcal{P}_i) + \text{ATR}_{\text{B}}(\mathcal{P}_i)\big). \quad (16)$$

Writing $\text{tool}(s)$ for the tool declared by step s, $\text{agent}(s)$ for the responsible agent, $\text{tools}(a)$ for the agent's declared tool set, $\mathcal{S}^{\text{tool}}_i = \{s : \text{tool}(s) \neq \varnothing\}$, and $v(s) = \mathbb{I}[\text{tool}(s) \in \text{tools}(\text{agent}(s))]$,

$$\text{ATR}_{\text{A}}(\mathcal{P}_i) = \begin{cases} 1 & \text{if } \mathcal{S}^{\text{tool}}_i = \emptyset, \\ \dfrac{1}{|\mathcal{S}^{\text{tool}}_i|} \sum_{s \in \mathcal{S}^{\text{tool}}_i} v(s) & \text{otherwise.} \end{cases} \quad (17)$$

Subscore B checks whether the plan as a whole declares the tools implied by the step text. A lexical *intent detector* matches keyword regexes against each step's instruction, objective, and expected-output fields and returns the set $\text{Intent}(s)$ of tools the text implies—for example `CodeInterpreterTool` for *execute / run / verify / compile*, or `CodeDocsSearchTool` for *retrieve / documentation / specification*. Patterns for frequent tools are mined from gold-step token statistics with recall validated on gold (99.6% for `CodeInterpreterTool`, 100% for `CodeDocsSearchTool`, 98.6% for `FirecrawlSearchTool`); patterns for sparse tools are derived from tool-name semantics. Letting $\mathcal{I}_i = \bigcup_{s \in \mathcal{S}_i} \text{Intent}(s)$ and $\mathcal{T}_i = \bigcup_{a \in \mathcal{A}_i} \text{tools}(a)$,

$$\text{ATR}_{\text{B}}(\mathcal{P}_i) = \begin{cases} 1 & \text{if } \mathcal{I}_i = \emptyset, \\ \dfrac{|\mathcal{I}_i \cap \mathcal{T}_i|}{|\mathcal{I}_i|} & \text{otherwise.} \end{cases} \quad (18)$$

Subscore B is intentionally plan-level rather than step-level: many gold steps contain teaching narratives such as "ask the learner to run X" that match a code-execution intent regex but do not require the agent itself to carry the tool (the learner does).

Aggregating intents across the plan avoids penalizing such narrative steps. Subscore A retains the strict step-level check as a backstop, so an agent can never call a tool it has not declared.

Profile-Grounded Personalization Reward. The main text treats the profile input abstractly through $\mathcal{C}(I_p)$ in Eq. 4. In the MAP-PPL implementation, I_p contains a structured tag list $\text{tags}(I_p)$ and a free-text self-description $\text{about}(I_p)$. We instantiate $\mathcal{C}(I_p) = \mathcal{K}_p \cup \mathcal{D}_p$ from the following two sources:

- $\mathcal{K}_p = \{\text{lower}(t) : t \in \text{tags}(I_p)\}$: lower-cased skill tags.
- $\mathcal{D}_p$: domain tokens of length ≥ 4 extracted from $\text{about}(I_p)$ using the regex `[A-Za-z][A-Za-z0-9_+#\-\.]+`, then filtered against a stopword list of (a) common English function words and (b) generic profession words such as *developer*, *engineer*, *experience*, *working*. $\mathcal{D}_p$ is further deduplicated against $\mathcal{K}_p$ so that a token already in the tag set is not double-counted.

Let $\text{Text}(\mathcal{P}_i)$ denote the lower-cased concatenation of every agent's role, goal, and backstory; every subtask's name and objective; and every step's instruction, objective, and expected output. The matched profile-signal set is

$$\text{match}(I_p, \mathcal{P}_i) = \{c \in \mathcal{C}(I_p) : c \text{ appears in } \text{Text}(\mathcal{P}_i)\}. \tag{19}$$

Substring matches use no word boundaries and no stemming, which is asymmetric in direction: the tag `json` matches a plan text containing `json.net`, but the tag `rails-activerecord` does not match a plan that only mentions `activerecord`. If $\mathcal{C}(I_p)$ is empty, Eq. 4 returns zero through its $\max(1, |\mathcal{C}(I_p)|)$ denominator.

Pedagogy Reward. We track the four Merrill-aligned teaching phases used in the main reward: ACTIVATION, DEMONSTRATION, APPLICATION, and INTEGRATION; let $\mathcal{M}$ denote this phase set. The Problem-centred principle is treated as a dataset-level by-construction condition rather than a per-plan check, since the programming query is given by the user and not produced by the model.

A lexical detector inspects the instruction, objective, and expected-output fields of each step and returns one phase in $\mathcal{M}$ or $\varnothing$ if no signature fires. Aggregating detector outputs over all steps gives $\text{phase}(\mathcal{P}_i)$ in Eq. 5. For ACTIVATION, lexical evidence alone is insufficient; the phase is counted only when the matched step also requires learner input:

$$\mathbf{1}_{\text{Act}}(\mathcal{P}_i) = \mathbb{I}[\exists s \in \mathcal{P}_i : \text{ActSig}(s) \wedge \text{LearnerInput}(s)], \tag{20}$$

where $\mathbf{1}_{\text{Act}}(\mathcal{P}_i) = \mathbb{I}[\text{ACTIVATION} \in \text{phase}(\mathcal{P}_i)]$. This implements Merrill's requirement that ACTIVATION actively elicits learner input, not merely mentions the learner. The reward thus takes values in $\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$. The reward does not enforce a fixed global order on phases. Merrill's principles form a cycle that can be iterated within a problem-centred plan, and only 16.6% of MAP-PPL gold plans satisfy the strict canonical prefix from Activation to Demonstration to Application to Integration; an order constraint would therefore mis-penalize a large fraction of pedagogically valid plans. We also do not include a plan-size envelope, since plan size is not a Merrill condition and earlier thresholded versions calibrated against gold percentiles risked rewarding superficial size mimicry.

Hard Feasibility Penalty. The hard penalty fires for failures that prevent the plan from being executed at all:

$$R_i^{\text{hard}} = -\lambda_{\text{hard}}\, \mathbb{I}\big[\text{SchemaInvalid}(\mathcal{P}_i) \vee \text{HasCycle}(G_i) \vee \text{InvalidToolCall}(\mathcal{P}_i)\big], \tag{21}$$

with $\lambda_{\text{hard}} = 10$. The three predicates cover non-recoverable execution failures:

- **Schema invalidity:** the model output is not parsable JSON, lacks required `agents`/`subtasks` blocks, or omits required step fields (`id`, `agent`, `instruction`).
- **Cyclic dependency:** the same acyclicity predicate as $\text{DAG}(\mathcal{P}_i)$ in the structural reward, escalated here from a soft signal to a hard gate.
- **Invalid tool call:** a step references a tool outside the fixed palette Π, or assigns a palette-valid tool to an agent that has not declared it. The palette Π contains code execution, documentation search, web search, file I/O, retrieval, and paper-search tools.

The value $\lambda_{\text{hard}} = 10$ is calibrated to the soft-reward scale: each soft reward lies in $[0, 1]$, so the un-z-scored soft contribution to R_i is at most $\sum_k w_k = 1$; with $\lambda_{\text{hard}} = 10$, no combination of soft scores can override an invalid plan, while

larger values would inflate the within-group standard deviation enough to compress z-scored soft advantages between valid candidates to near zero. Earlier versions of this gate also penalized plans with zero learner-interaction steps and plans with no code-tool step; both were removed after we observed entire rollout groups failing the same binary check, which collapsed group variance and erased the GRPO advantage signal. The learner-interaction requirement is now expressed softly through the ACTIVATION phase of R^{ped}, and the code-tool requirement through ATR_B of R^{struct}.

Group Normalization and Dynamic Reweighting. Within a rollout group $\mathcal{G}_I$ of candidates sharing the same input $I = (I_q, I_p)$, each soft axis $k \in \{\text{struct}, \text{pers}, \text{ped}\}$ is z-scored:

$$\bar{R}_i^k = \begin{cases} \dfrac{R_i^k - \mu_I(R^k)}{\sigma_I(R^k) + \varepsilon} & \text{if } \sigma_I(R^k) > \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

where μ_I and σ_I are the within-group mean and standard deviation, ε is a numerical stabilizer, and the τ threshold prevents zero-variance groups from blowing up the z-score. The hard penalty R_i^{hard} is never z-scored, so it preserves its absolute magnitude even when an entire rollout group is invalid.

Soft-axis weights $(w_{\text{struct}}, w_{\text{pers}}, w_{\text{ped}})$ are initialized to $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. At each training step, we maintain an exponential moving average (EMA) of the raw batch mean per axis,

$$m_k^{(t)} = (1-\alpha)\, m_k^{(t-1)} + \alpha\, \overline{R^k}_{\text{batch}}, \tag{23}$$

with $\alpha = 0.10$. During a warmup of the first $T_{\text{warm}} = 50$ steps the EMAs are tracked but the weights are not yet updated. After warmup, every $T_{\text{refresh}} = 100$ steps we apply a clipped multiplicative update against a reference target ρ:

$$\begin{aligned} \text{gap}_k &= \rho - m_k, \\ w_k' &= \text{clip}\big(w_k \cdot \exp(\eta_w \cdot \text{gap}_k),\ w_{\min},\ w_{\max}\big), \\ w_k &\leftarrow w_k' / \textstyle\sum_j w_j', \end{aligned} \tag{24}$$

with $\rho = 0.40$, $\eta_w = 0.30$, $w_{\min} = 0.05$, $w_{\max} = 0.70$. Axes whose EMA lags the reference receive an upward push and those that exceed it are pushed down; the clip-and-renormalize step prevents any axis from collapsing to zero (which would terminate gradient flow on that axis) or monopolizing the composite loss. The final per-rollout reward fed back to GRPO is

$$R_i = \sum_{k \in \{\text{struct},\text{pers},\text{ped}\}} w_k\, \bar{R}_i^k + R_i^{\text{hard}}, \tag{25}$$

which the GRPO loop further standardizes within the same rollout group to produce the advantage A_i used by the clipped surrogate in Section 3.3.

A.2 MAP-PPL Dataset Analysis

This appendix collects the figures and tables deferred from Section 4. All numbers are computed on the released 3,043-instance benchmark with the same scripts that produced the HTML analysis report shipped with the dataset. The subsections below mirror the five claims of the main text (construction integrity, planning complexity, profile-conditioned structural personalization, pedagogical scaffolding, and split/quality/bias control), with per-intent, per-language, and dataset-comparison panels added as auxiliary evidence.

A.2.1 Construction funnel and rejection reasons

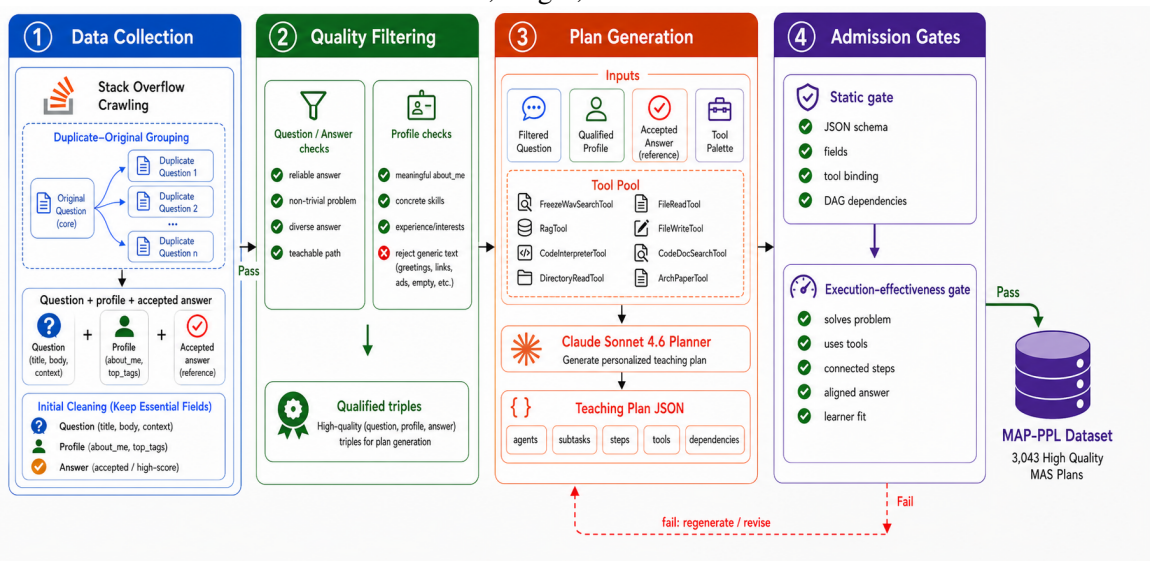
Figure 5 shows the per-stage admission funnel that yields the released MAP-PPL set. Starting from the raw Stack Overflow duplicate-question groups we collect, we successively discard (a) question groups whose accepted/high-vote answer is too short, code-only, or unverifiable; (b) profiles that contain no concrete personalization signal beyond a username; (c) generated plans that fail any of the static-gate checks listed in Section 4.1; and (d) plans that pass the static gate but fail the LLM execution-effectiveness gate. The first three columns of Table 3 attribute the discarded plans at each gate to the most common failure reason; the fourth column reports the mean number of regeneration attempts before admission, which we cap at three.

The dominant failures are weak personalization (a profile is mentioned in the plan only at the surface), shallow decomposition (fewer than three substantive steps before the first `validate` step), and dependency mismatch (an `execution_order` item with no matching `depends_on` chain). Schema-level failures (invalid agent reference, cycle, loop-step referring to an unknown node) account for less than 5% of static-gate rejections, consistent with the 100% structural validity rate on the released set.

A.2.2 Plan-generation prompt template

Figure 6 shows the abridged prompt used to synthesize a MAP-PPL plan from a Stack Overflow question, learner profile, accepted answer, and tool palette. The implemented prompt is longer because it includes field-level length budgets, loop rules,

; height;



height

Figure 5: Construction funnel for MAP-PPL: raw duplicate-question groups → reliable-answer groups → profile-qualified triples → generated plans → static-gate pass → LLM-gate pass → released set.

Table 3: Rejection reasons across the two-stage admission gate. *Reject*: percentage of plans entering the gate that the gate discards. *Reattempts*: mean regenerations before admission (capped at three).

Gate	Top failure reason	Reject	Reattempts
Static	invalid `execution_order`	11.3%	0.9
Static	unknown `depends_on` target	7.2%	0.7
Static	cycle / loop refers to unknown node	2.4%	0.4
LLM	weak / surface personalization	14.6%	1.3
LLM	shallow decomposition	10.8%	1.1
LLM	misalignment with accepted answer	8.1%	0.9
LLM	implausible tool usage	3.7%	0.5

and examples of valid plan shapes; the figure preserves the operative structure: answer-grounded target extraction, profile-conditioned decisions, minimal agent/tool selection, strict JSON generation, and post-generation self-checks.

A.2.3 Input lengths, profile depth, and profile pairing

The basic input–side statistics show that query character length has mean 592, median 532, and maximum 2,340; 95% of profiles list four or five technology tags; and 43.9% of questions have a single profile while 56.1% have between 2 and 6. The right-skewed query-length distribution and the heavy concentration of full-tag profiles together explain why the PAD adapter benefits from explicit structural supervision: most queries are long enough and most profiles dense enough that a generic template will mis-route the early scaffolding decision.

A.2.4 Executability diagnostics: DAG topology, tool bindings, and runtime audit

Figure 9 shows the marginal distributions for the four structural axes used in the headline table. Plans are concentrated around 3 agents, 4 subtasks, and 10 steps, with right tails extending to 4, 6, and 23 respectively. The fraction of `human_input`-tagged steps clusters tightly around 0.55, confirming the interactive teaching pattern reported in the main text. Loop structures are present in 80.6% of plans; the loop maximum-iteration parameter is 3 for 83% of loops and 2 otherwise, so the planner does not synthesize unbounded retries.

Beyond plan-size marginals, MAP-PPL supplies non-trivial DAG topology rather than linear checklists. Table 4 summarizes the graph-level metrics that justify the "multi-agent planning" label for the dataset: edge density is well above zero but below unity, the mean per-step out-degree is below one with a long tail (a small fraction of fork steps fan out to as many as four downstream steps), the critical-path-to-total-steps ratio is roughly one half (so plans are not single long chains), and 58.5% of dependency edges cross agent boundaries – the structural fingerprint of a real handoff. Figure 7 gives a compact executability overview, while Figure 8 reports the empirical distributions of edge density, fan-in/fan-out, and parallelizable layer width, together with a graph-motif bar chart that decom-

MAP-PPL Plan Generation Prompt (Abridged)

System prompt: Generate a personalized multi-agent plan—a strict JSON specification of agents, subtasks, steps, and dependencies—for a learner solving a Stack Overflow question. The plan will be consumed by a CrewAI-style runtime.

Inputs: `query`, `learner.self_description`, `learner.skills`, accepted answer, and the declared tool pool.

Tool pool: `FirecrawlSearchTool`, `RagTool`, `CodeInterpreterTool`, `DirectoryReadTool`, `FileReadTool`, `FileWriterTool`, `CodeDocsSearchTool`, `ArxivPaperTool`.

Core instructions:

- Use the accepted answer only to extract the destination concept and success criteria.
- Infer the learner's starting point, explanation bridge, and interaction style from profile text and tags.
- Decompose the gap into observable subtasks, instantiate the minimal specialized agent/tool set, and add tool calls only when the step needs external capability.
- Emit only valid JSON; every dependency must point to a step whose output is used downstream.

```
Output schema:
{
  "output": {
    "agents": [{"agent_role", "goal", "backstory", "tools"}],
    "subtasks": [{
      "id", "name", "subtask_objective",
      "steps": [{
        "id", "agent", "objective", "instruction", "tool",
        "requires_human_input", "expected_output", "depends_on"
      }]
    }],
    "execution_order": [
      "S1-1",
      {"loop": {"steps": ["S2-1","S2-2","S2-3"],
                "condition": "S2-2.correct == false",
                "max_iterations": 3}},
      "S2-4"
    ]
  }
}

Self-check:
JSON parses; all steps are scheduled; depends_on targets resolve;
the graph is acyclic; tools are in-palette and agent-authorized;
replacing the learner profile would require changing structure or framing.
```

Figure 6: Abridged plan-generation prompt used to synthesize MAP-PPL plans. The released data are generated with the full prompt, which expands the field-level style rules, loop constraints, and valid plan-shape examples while preserving this structure.

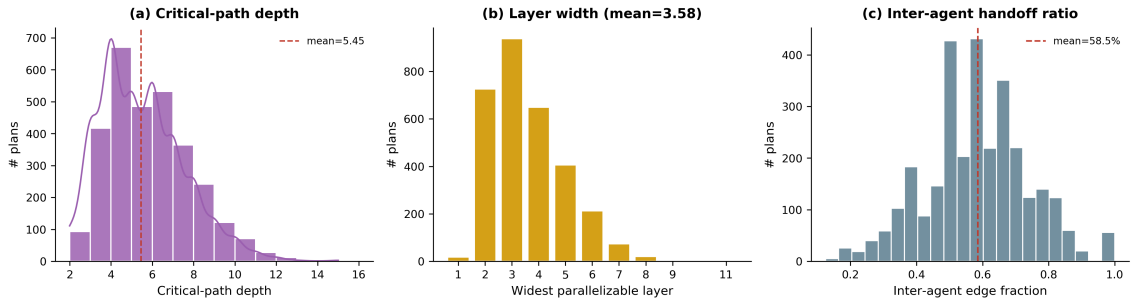


Figure 7: DAG executability in **MAP-PPL**. The released plans combine prerequisite depth, parallelizable layer width, and inter-agent dependency edges, supporting executable multi-agent workflows rather than flat teaching checklists.

poses each plan into chain, fork, join, and feedback-loop motifs, and an inter-agent handoff heatmap over the four high-level role families.

Schema and runtime checks. Executability is enforced before release. All 3,043 plans pass the static structural checker: 0 contain an unknown agent reference, 0 contain an unknown `depends_on` target, 0 contain an unknown step in `execution_order`, 0 omit a step from `execution_order`, 0 contain a cycle, and 0 contain a loop reference to an unknown node. The loader normalizes historical field variants for loop specifications and then executes the released plans with a CrewAI-style teacher–learner simulator. The runtime executes bounded loops and code-tool calls, and every released plan completes with `status=ok`, `succeeded=true`, and 0 failed steps.

Tool bindings. Tool use is compact and programming-specific. Although the schema declares eight possible tools, only five appear in the final plans; `CodeInterpreterTool` and `CodeDocsSearchTool` together cover 99.6% of tool calls, while search and file-writing tools form a small long tail. The static gate also checks that each step's tool call is drawn from the declared palette and is authorized for the responsible agent, so the DAG edges, agent assignments, and tool bindings are jointly executable rather than independently valid.

A.2.5 Complexity by intent

Table 5 reports the per-intent structural averages used by the intent-conditioned slices of our evaluation. LEARNING queries demand the largest plans (3.5 agents, 11.92 steps, longest path 6.92) while ERRORS and API_CHANGE produce the shortest ($\approx$ 9.5 steps, longest path 5.3–5.4); the average number of human-input steps and the loop rate also rise on diagnosis-heavy intents (REVIEW, DISCREPANCY), confirming the dataset captures intent-specific pedagogy.

Table 4: DAG topology metrics on the 3,043 released plans. *Edge density*: $|E|/|S|(|S|-1)$. *Out-degree*: per-step out-degree across the plan. *Critical/total ratio*: longest DAG path / total step count. *Inter-agent ratio*: fraction of dependency edges whose two endpoints belong to different agents.

Metric	Mean	Median	P90	Max
Edge density	0.08	0.08	0.11	0.21
Mean out-degree	0.76	1.00	1.00	5.00
Max fan-out per plan	1.73	2.00	2.00	5.00
Max fan-in per plan	1.69	2.00	2.00	5.00
Max layer width	3.58	3.00	6.00	11.00
Critical / total step ratio	0.53	0.50	0.78	1.00
Inter-agent edge ratio	0.59	0.57	0.80	1.00

Table 5: Per-intent structural averages in **MAP-PPL**. $|\mathcal{A}|$: agents; $|\mathcal{T}|$: subtasks; $|\mathcal{S}|$: steps; $L_{\max}$: longest DAG path; *HI*: average number of `human_input` steps per plan; *Loop%*: fraction of plans with ≥ 1 loop.

Intent	n	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{S}\rvert$	$L_{\max}$	*HI*	*Loop%*
CONCEPTUAL	1488	2.90	4.03	10.54	5.16	5.73	69.0
API_USAGE	1114	2.93	3.95	10.31	5.77	5.63	93.0
DISCREPANCY	172	2.85	3.85	9.98	5.49	5.60	90.0
REVIEW	129	2.81	3.91	10.32	5.88	6.19	93.0
ERRORS	70	2.84	3.66	9.54	5.39	5.39	83.0
API_CHANGE	58	2.90	3.78	9.48	5.34	4.97	81.0
LEARNING	12	3.50	4.50	11.92	6.92	5.83	92.0

A.2.6 Agent role families and tool usage

Figure 10 reports the agent-side diversity statistics summarized in the main text. The top-15 role names (panel a) are dominated by language-specific validator/retriever pairs, which is the fine-grained surface form of the four high-level role families (panel b: TUTOR, VALIDATOR, RETRIEVER, DEBUGGER, plus minor

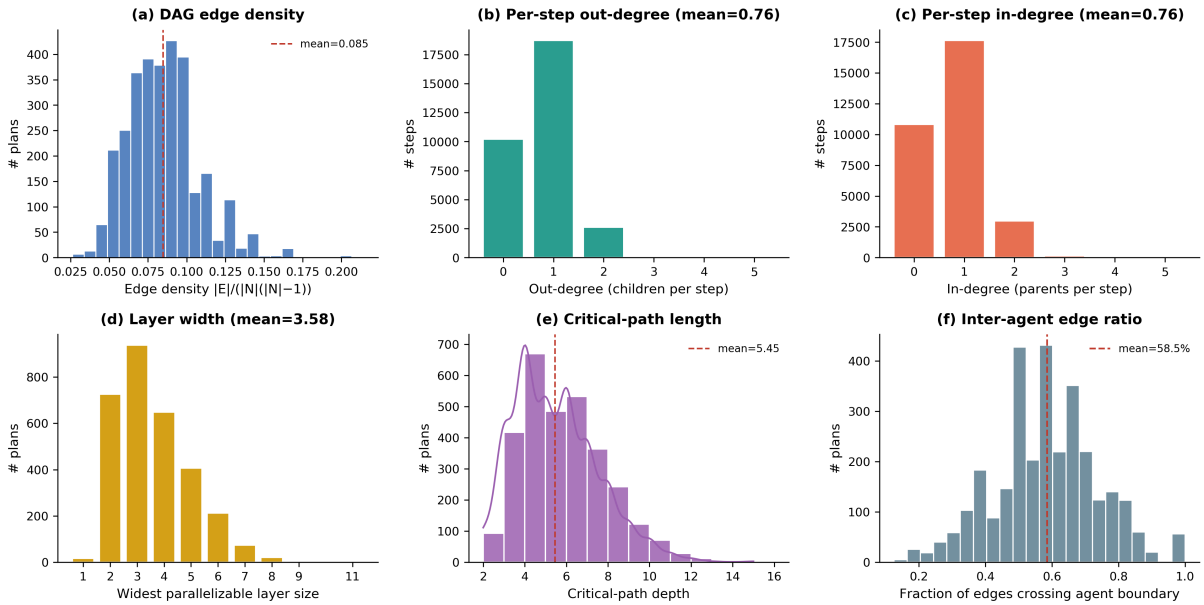


Figure 8: Dependency-graph topology in **MAP-PPL**: edge density, fan-in/fan-out, parallelizable layer width, motif counts (chain/fork/join/loop), and the agent-family handoff heatmap. The last panel confirms that handoffs concentrate on the TUTOR→VALIDATOR and RETRIEVER→TUTOR transitions rather than staying inside a single agent.

categories). The tool palette is intentionally small (panel c): `CodeInterpreterTool` and `CodeDocsSearchTool` cover nearly all tool calls. We use this empirical tool distribution as the ground truth against which the GRPO agent–tool relevance reward ATR is calibrated.

A.2.7 Cross-profile personalization

The cross-profile analysis substantiates the "scaffold-stable, surface-variable" personalization pattern reported in the main text. Across the 1,777 within-question profile pairs, agent role names and subtask names diverge sharply (median Jaccard 0.20 and 0.25), while role families and tools remain near 1; in other words, personalization changes *who* the planner instantiates but not the pedagogical scaffold. Profile grounding is also strong: 95.1% of plans mention at least one declared profile skill, with mean skill-hit ratio 0.43 and median 0.40. Role-level plan divergence ($1 - \text{Jaccard}$) remains high across all seven intents (0.65–0.87), indicating that profile-conditioning does not collapse to a single template even for the dominant CONCEPTUAL and API_USAGE categories.

To separate structural personalization from surface keyword hits we additionally run a profile-shuffled control: for every within-question pair we re-pair each profile with a plan generated for a different question that shares the same intent, and re-evaluate the four divergence axes (Table 6). The skill-hit ratio collapses from 0.43 to 0.10 and the agent-role Jaccard rises from 0.32 to 0.71, confirming that the observed divergence in the genuine pairs is driven by the profile signal rather than by sampling variance. The structural diagnostics agree: the dependency-edge Jaccard rises by 24 points and the validation-step prerequisite depth varies in only 4% of shuffled pairs (vs. 42% in the genuine pairs).

Table 6: Genuine vs. shuffled profile pairs on the 1,777 within-question pairs. *Skill-hit*: fraction of profile-declared skills mentioned in the plan. *Agent / Edge J.*: role-name and dependency-edge Jaccard between the two plans. *Val-depth* Δ: fraction of pairs whose pre-VALIDATE prerequisite chain length differs.

Pair type	Skill-hit	Agent J.	Edge J.	Val-depth Δ
Genuine within-question	0.43	0.32	0.46	42%
Profile-shuffled control	0.10	0.71	0.70	4%

Table 7 walks through a single concrete example: an identical `TypeError: 'NoneType' object is not subscriptable` query is paired with (a) a junior front-end developer profile that lists JavaScript and React only, and (b) a senior data engineer profile that lists Python, Pandas, and Airflow. The two MAP-PPL plans agree on the high-level TUTOR→VALIDATOR scaffold and on the `CodeInterpreterTool` / `CodeDocsSearchTool`

tool pair, but diverge in (i) the agent roster (a Python-debugging agent vs. a JS-debugging agent), (ii) the prerequisite depth before VALIDATE (4 vs. 2 steps), and (iii) the chosen worked-example domain (Pandas DataFrames vs. React state hooks). This pattern is exactly the "scaffold-stable, surface-variable" behaviour we expect MAP-PPL to teach.

A.2.8 Pedagogical phase coverage, order, and instructional methods

Figure 12 reports the pedagogical phase coverage at the plan and step level (panel a) and the coverage of nine canonical instructional methods (panel b). To match the reward formulation in Section 3.3, we report the four core detector labels as Merrill-aligned phases: ACTIVATION (PROBE), DEMONSTRATION (RETRIEVE–DEMONSTRATE), APPLICATION (APPLY), and INTEGRATION (VALIDATE). These four phases are present in 100.0%/98.1%/99.7%/100.0% of plans respectively, with the optional diagnostic labels FEEDBACK and CONSOLIDATE at 89.4% and 96.6%. At the step level, 70.4% of steps map to ACTIVATION, 33.3% to DEMONSTRATION, 62.6% to APPLICATION, and 60.1% to INTEGRATION. Only 16.6% of gold plans achieve the strictly ordered four-phase prefix, which supports the phase-coverage reward in Section 3.3 rather than a fixed global-order constraint.

Phase-coverage detector versions. An earlier draft of this paper reported CONSOLIDATE coverage as 86.3% using a v1 lexical detector that required the literal keyword `consolidate` in the step instruction; the v2 detector used throughout the current paper also accepts paraphrases (e.g. "recap", "summarize", "key takeaways") and raises the coverage to 96.6%. The v1 number is not used anywhere in the released paper; we mention the discrepancy explicitly so that downstream readers can reproduce both numbers from the released phase-detector dictionaries. All phase-level claims (main text and appendix) are computed under v2, while the GRPO reward itself uses only the four Merrill-aligned phases above.

Phase position and order, not just presence. Near-100% phase coverage can mask templated plans that touch each phase exactly once in a fixed slot, so we additionally analyse the *position* of each phase within the `execution_order`. Phase positions follow this canonical ordering on average – the median normalized position of ACTIVATION is 0.16 (first quartile of steps) and that of INTEGRATION is 0.83 (last quartile) – while still exhibiting substantial variance: only 16.6% of plans realise the strict prefix and a Sankey decomposition of phase-to-phase transitions (deferred to the dataset card) shows that plans interleave APPLICATION and INTEGRATION loops rather than running each phase exactly once.

A.2.9 Holistic quality audit

Executability audit. The schema, DAG, tool-binding, and runtime execution checks are consolidated in Appendix A.2.4. This section focuses on complementary semantic quality checks.

Manual audit on a stratified sample. Structural validity certifies JSON/DAG soundness but does not, by itself, certify that a plan is pedagogically meaningful, that the profile is used non-superficially, or that the chosen tools and dependencies are semantically appropriate. We therefore complement the LLM execution-effectiveness gate with a human audit on a stratified sample of $N{=}200$ plans drawn proportionally over the seven intents and the four agent-count strata. Each sampled plan is labelled independently by two annotators along five binary criteria: (C1) schema-valid *and* all `depends_on` edges semantically justified, (C2) agent roster minimal but sufficient, (C3) tool calls plausible for the assigned step, (C4) profile adaptation is structural rather than surface-only, and (C5) the plan covers the core concepts of the accepted Stack Overflow answer. Table 8 reports per-criterion agreement rates and Cohen's κ, together with the agreement between the LLM gate decision (admit/reject) and the human consensus label, which we use to bound the residual judge bias inherited by the supervision set. The agreement between the LLM gate and human consensus reaches 0.87 across the audited sample, with the largest gap on C4 (profile-adaptation depth) where the LLM gate is slightly more lenient than human raters.

Failure modes. The most common failures on the audited sample concentrate on C2 (a redundant retriever agent that the dependency graph never calls into) and C4 (the plan paraphrases the profile in the `about_me` section but otherwise reuses the default scaffold). Both are surfaced as structured penalties during GRPO: unused agents contribute to a step-utilization sub-reward, and shallow profile adaptation is penalised by the personalization proxy described in Section 3.3.

Table 7: Matched contrastive case: same query, two profiles. *Stable* indicates plan components shared between both profiles; *Changed* indicates components rewritten in response to the profile.

Plan component	**Profile A (junior front-end)**	**Profile B (senior data eng.)**
Stable: role family	TUTOR, VALIDATOR	TUTOR, VALIDATOR
Stable: tool palette	`CodeInterpreter`, `Docs`	`CodeInterpreter`, `Docs`
Changed: agent roster	React-Debugger, JS-Tutor	Pandas-Debugger, Py-Tutor
Changed: probe focus	null-handling in JSX props	`None` in DataFrame indexing
Changed: worked example	`useState` hook null-guard	Pandas `loc` / `merge` guard
Changed: prerequisite depth	2 steps before VALIDATE	4 steps before VALIDATE
Changed: feedback loop	1 iteration of PROBE-APPLY	2 iterations with CONSOLIDATE step

Table 8: Manual audit on a stratified N=200 sample. *Pass*: percentage of audited plans for which both annotators marked the criterion satisfied. κ: Cohen's κ between the two annotators. *LLM agr.*: agreement between the LLM execution-effectiveness gate decision and the human consensus label.

Criterion	**Pass**	κ	**LLM agr.**
C1 Schema + dependency semantics	96.5%	0.81	0.94
C2 Minimal-sufficient agent set	92.0%	0.74	0.88
C3 Tool plausibility	94.0%	0.78	0.91
C4 Structural profile adaptation	88.5%	0.69	0.81
C5 Concept coverage vs. answer	90.5%	0.72	0.86
Overall (any criterion fails)	83.5%	0.71	0.87

A.2.10 Comparison to existing datasets

Table 9 positions MAP-PPL against the two dataset families it is most often compared with at review time: agent-trajectory benchmarks for tool use and multi-step planning (AgentBank, AgentGym, workflow-style datasets), and educational / tutoring datasets (instruction tuning for tutoring, conversation traces with learner moves). The axes are exactly the ones our main-text claims commit to: explicit learner-profile conditioning, multi-agent plan schema rather than single-trajectory tool calls, DAG-typed dependencies, a fixed tool palette with execution constraints, pedagogical phase scaffolding, and a structural-plus-execution admission gate. To our knowledge, no prior public dataset combines all six axes; MAP-PPL's contribution is to supply the supervision required for profile-conditioned multi-agent planning in programming education rather than to compete on raw size with general agent benchmarks.

A.3 Implementation Details

This appendix collects the implementation details for the experimental setup in Section 5.

Baseline selection. The three baseline groups are chosen to separate three possible sources of planning ability. Frontier LLM planners test whether strong general-purpose models can directly emit personalized programming-learning MAS plans without task-specific training. Generic MAS framework planners test whether existing agent-roster and coordination frameworks are sufficient once they are given the same learner profile and tool pool. Agentic workflow planners test whether systems that explicitly generate or compose agent/task workflows can transfer to personalized tutoring-plan generation. We do not include educational-agent systems such as EduPlanner (Zhang et al., 2025b) and GenMentor (Wang et al., 2025) in the comparison because they output tutoring content or learning-session outlines rather than MAS orchestration plans, making them incompatible with our plan-level evaluation.

Baseline instantiation. All open-source or framework-based baselines and PersonalPlan use Qwen3-32B-Instruct as the shared backbone. Closed-source LLM baselines are prompted once to emit the full MAP-PPL plan. AutoGen and AutoAgents are instantiated with the shared tool pool and converted into the same JSON plan schema. AIPOM is evaluated through its single-shot agent-aware DAG planner, and AFlow through its operator-composition planning step over the same available tools.

Shared schema and tool pool. Every method is required to output the same plan schema with agents, subtasks, steps, dependency edges, and an execution order. Tool calls are restricted to the fixed MAP-PPL palette so that structural validity, dependency completeness, and agent–tool relevance are comparable across methods.

PersonalPlan variants. The main model uses the two-stage training pipeline from Section 3: PAD and SDP LoRA adapters for hierarchical SFT, joint alignment to reduce PAD–SDP exposure mismatch, and Reward-Adaptive GRPO initialized

Table 9: MAP-PPL versus representative agent-trajectory and educational datasets along the six axes that motivate the benchmark. ✓ / × indicate whether the dataset, as publicly released, supplies the feature; "partial" marks features that the dataset provides only at coarser granularity than MAP-PPL.

Dataset family	**Profile**	**Multi-agent**	**DAG deps.**	**Tool palette**	**Pedagogy**	**Exec. gate**
Agent-trajectory (AgentBank, AgentGym, . . .)	×	partial	linear trace	✓	×	partial
Workflow-style planning (AFlow, AOP, . . .)	×	✓	DAG	✓	×	✓
Tutoring dialogue (instruction + learner moves)	partial	×	×	×	partial	×
Programming Q&A (SO-derived corpora)	×	×	×	×	×	partial
MAP-PPL (ours)	✓	✓	DAG	✓	✓	✓

from the SFT model. Ablation variants remove or replace one component at a time while preserving the same data splits, schema, tool pool, and evaluation protocol.

A.3.1 Shared Baseline Plan-Generation Prompt

All planner baselines use the following shared prompt template. Method-specific wrappers only adapt the prompt to the corresponding API or framework interface; the task definition, tool palette, schema, and output constraints are kept identical.

A.4 Evaluation Metrics

For reproducibility, we make every metric's predicate explicit: each formula is followed by a numbered list of the conditions under which it evaluates positively, so no metric depends on an unexpanded function name. Throughout, $\mathcal{P}$ denotes a candidate plan, $\mathcal{P}^\star$ the gold plan, q the query, I_p the learner profile, $\mathcal{S}/\mathcal{E}$ the step set / prerequisite edge set, $\mathcal{R}$ the set of executed runs, and $\mathbb{I}[\cdot]$ the indicator function. Numerical anchors $\{1, 3, 5\}$ in LLM-judge scores follow KELE-style 5/3/1 rubrics.

(a) Static plan evaluation: generation robustness and executable structure. All methods are evaluated under the same bounded retry-until-valid protocol. Let N be the number of test instances, A_m the total number of model generation calls made by method m before the final accepted set is assembled, and H_m the number of query–profile instances whose plans still fail the automatic audit after three same-query reruns and therefore require deterministic post-hoc repair.

• **Attempts.**

$$\mathrm{Attempts}(m) = \frac{A_m}{N}.$$

A value of 1.0 means every instance is accepted on the first generation call; larger values indicate more retry cost. This metric is lower-is-better.

• **Repair Rate.**

$$\mathrm{RepairRate}(m) = \frac{H_m}{N}.$$

A repair is counted once per query–profile instance, not once per failed generation call: if any valid plan is obtained within the three same-query reruns, the instance contributes 0 to H_m; if all reruns still fail the audit and the instance is recovered by deterministic post-processing such as JSON escaping, schema-field completion, or tool-declaration filling, it contributes 1. This metric is lower-is-better. The schema, tool-permission, and topological checks used for acceptance are retained as audit logs rather than headline columns because the retry protocol intentionally saturates final validity.

• **Tool-Binding Quality (TBQ).** TBQ evaluates whether tool calls are semantically appropriate for the agent role and the current step, while also rewarding broad tool deployment across the plan. Let $\mathcal{T} = \{s \in \mathcal{S} : \mathrm{tool}(s) \neq \varnothing\}$ be the tool-bound steps and $c = |\mathcal{T}|/|\mathcal{S}|$ be tool coverage. For text fields x_s, y_s, define

$$I_{XY} = \frac{1}{|\mathcal{T}|} \sum_{s \in \mathcal{T}} \mathbb{I}[\cos(\mathrm{enc}(x_s), \mathrm{enc}(y_s)) > \theta],$$

where $\mathrm{enc}(\cdot)$ is the sentence encoder (all-MiniLM-L6-v2 in our implementation) and $\theta = 0.10$. For each step, role text comes from the responsible agent description, subtask text from the subtask name and objective, step text from the instruction fields, and tool text from the tool's natural-language description. We instantiate the three pairs as role–subtask, tool–role, and tool–step:

$$\mathrm{TBQ}(\mathcal{P}) = \frac{I_{\mathrm{AS}} + c\big(I_{\mathrm{TA}} + I_{\mathrm{TS}}\big)}{3}.$$

Here I_{AS} checks whether the assigned agent role matches the subtask, I_{TA} checks whether the tool matches that role, and I_{TS} checks whether the tool matches the step instruction. The two tool-specific terms are multiplied by c, so a plan cannot score

highly by making only one correct tool call. TBQ is reported only for plans with at least one tool-bound step.

• **Topology Similarity (TS).** TS compares the candidate dependency graph $G = (\mathcal{S}, \mathcal{E})$ with the gold graph $G^\star = (\mathcal{S}^\star, \mathcal{E}^\star)$ using graph edit distance, then applies a capped compactness factor. Let $n = |\mathcal{S}| + |\mathcal{E}|$, $n^\star = |\mathcal{S}^\star| + |\mathcal{E}^\star|$, and $\kappa = 2.5$:

$$B(G, G^\star) = \max\left(0,\, 1 - \frac{\mathrm{GED}(G, G^\star)}{\max(n, n^\star)}\right).$$

$$C(G, G^\star) = \min\left(\kappa,\, \frac{n^\star}{\max(1, n)}\right).$$

$$\mathrm{TS}(\mathcal{P}, \mathcal{P}^\star) = \min(1, B(G, G^\star)\, C(G, G^\star)).$$

The base term B is the normalized topology overlap, while C rewards candidates that recover the reference dependency structure with fewer steps. The cap prevents very small degenerate plans from receiving unbounded compactness credit. We compute GED with unit-cost graph edit operations and a 2-second timeout, treating node labels as interchangeable.

(b) Static plan evaluation: personalization and pedagogy. For each subdimension d, $J_d(\cdot)$ returns an anchored score $r_d \in \{1, 3, 5\}$ governed by the rubric: r_d=5 when all positive anchors hold, r_d=3 when one anchor fails, r_d=1 when two or more fail.

• **Personalization (Pers.).** SkillMatch / GoalOrientation / BackgroundAdaptation, each from an independent judge call:

$$\mathrm{Pers.} = \frac{1}{4}\left(\frac{r_{\mathrm{sm}} + r_{\mathrm{go}} + r_{\mathrm{ba}}}{3} - 1\right) \in [0, 1].$$

• **Pedagogy (Ped.).** Combines four KELE-style sub-axes: r_{PRR}, NDAR, r_{IAR}, and a rule-based phase coverage

$$\mathrm{SPR} = \frac{1}{|\mathcal{M}|} \sum_{\phi \in \mathcal{M}} \mathbb{I}[\phi \in \mathrm{phase}(\mathcal{P})]$$

over the four Merrill-aligned phases used in Eq. 5. Aggregation:

$$\mathrm{Ped.} = \frac{1}{4}\Bigg(\frac{r_{\mathrm{PRR}} - 1}{4} + \mathrm{NDAR} + \mathrm{SPR} + \frac{r_{\mathrm{IAR}} - 1}{4}\Bigg).$$

(c) Static plan evaluation: profile-counterfactual probes.

• **Profile-Variance Score (PVS).** For each query with $k_q \geq 2$ profile variants ($|\mathcal{Q}_{\geq 2}|$=97),

$$\mathrm{PVS}(q) = 1 - \binom{k_q}{2}^{-1} \sum_{i<j} \mathrm{TopoSim}(\mathcal{P}_q^{(i)}, \mathcal{P}_q^{(j)}),$$

and $\mathrm{PVS} = \mathbb{E}_q[\mathrm{PVS}(q)]$. Here TopoSim is the normalized graph-overlap base used by TS, without the compactness factor because both arguments are generated plans.

• **Profile–No-Profile Gap (PNG).** With $\tilde{I}_p$ a profile sampled uniformly from other test profiles,

$$\mathrm{PNG} = \frac{1}{N} \sum_i \Big[\, \mathrm{Pers.}\Big(\mathrm{gen}(q_i, I_p^{(i)}),\, I_p^{(i)}\Big) - \mathrm{Pers.}\Big(\mathrm{gen}(q_i, \tilde{I}_p^{(i)}),\, I_p^{(i)}\Big) \Big].$$

(d) Deterministic post-execution structure.

• **Structural Compactness Score (SCS).** SCS is computed from the generated plan and its execution log without an LLM judge. For each executed run r, let L_r be the ordered execution log, P_r the corresponding plan, and $n_r = |L_r|$. We first compute four raw quantities:

$$\mathrm{triv}(r) = \frac{|\{e \in L_r : |\mathrm{strip}(\mathrm{agent_output}(e))| < 100\}|}{\max(1, n_r)}.$$

$$\mathrm{subt}(r) = \frac{|\{\mathrm{subtask_id}(e) : e \in L_r\}|}{\max(1, n_r)}.$$

$$\mathrm{tool}(r) = \begin{cases} \dfrac{|D(P_r) \cap U(L_r)|}{|D(P_r)|}, & |D(P_r)| > 0, \\ 1, & |D(P_r)| = 0, \end{cases}$$

where $D(P_r)$ is the set of non-null tools declared by plan steps, and $U(L_r)$ is the set of tool names extracted from agent outputs with deterministic invocation patterns such as `Tool: X`, `using X`, or `invoke X`. For each method m, we average these raw quantities over its executed runs to obtain $\bar{n}_m$, $\bar{t}_m$, $\bar{u}_m$, and $\bar{\rho}_m$. Let $\mathcal{M}$ be the compared method set. The four normalized SCS subdimensions are

$$C_m = \frac{\max_{j \in \mathcal{M}} \bar{n}_j - \bar{n}_m}{\max_{j \in \mathcal{M}} \bar{n}_j - \min_{j \in \mathcal{M}} \bar{n}_j}, \qquad K_m = 1 - \frac{\bar{t}_m}{\max_{j \in \mathcal{M}} \bar{t}_j}.$$

$$G_m = \frac{\bar{u}_m}{\max_{j \in \mathcal{M}} \bar{u}_j}, \qquad T_m = \frac{\bar{\rho}_m}{\max_{j \in \mathcal{M}} \bar{\rho}_j}.$$

If a denominator is zero, the corresponding normalized term is set to 1 for all methods. The reported score is the equally weighted mean:

$$\mathrm{SCS}(m) = \tfrac{1}{4}(C_m + K_m + G_m + T_m).$$

Here C_m measures compactness, K_m cleanness against trivial outputs, G_m concentration of unique subtasks per step, and T_m realized tool usage. Higher SCS indicates a more compact, cleaner, more focused, and more tool-realized execution trace under the same runtime.

(e) Programming-specific post-execution metrics. **CER**: fraction of code-bearing steps whose output contains real `stdout/stderr` rather than speculative narration. **CCR**: fraction of executed code steps whose actual output is functionally equivalent to `expected_output` under GPT-5 judge. **LST**: fraction of runs (excluding *perfect-from-start* IQ degenerates) in which the student's final code attempt is functionally equivalent to the accepted answer. **MLC**: per-language CER/CCR breakdown over fourteen languages whenever $|\mathcal{S}_\ell^{\text{code}}| \geq 5$.

(f) Aggregates and judge consistency.

• **Satisfaction (Sati.).** Per-pair score $\pi_i^{(m)}$ awards $+\frac{1}{2}$ for each AB / BA candidate vote and $+\frac{1}{2}$ for AB$\neq$BA (treating order instability as a tie). The cross-judge score is the multiplicative product of per-judge candidate-win rates w_m, which penalizes asymmetrically:

$$\text{Sati.} = \prod_m w_m.$$

• **Judge Consistency (JCC).** Krippendorff α across $M{=}3$ judges with nominal-level distance; threshold $\alpha \geq 0.5$ is required for Sati. to be reportable.

A.5 Evaluation Metric Prompt Templates

Inspired by VAIAGE (Liu et al., 2025), we adopt a multi-faceted evaluation approach, combining LLM-based quality assessment with quantitative structural metrics.

A.5.1 LLM-based Scoring (0-1 Scale)

We use LLMs as proxy judges for plan-level personalization and pedagogy, dialogue-level pedagogy, and profile-conditioned satisfaction. Each score is elicited via an anchored prompt with explicit criteria and a strict JSON output schema. Specifically, the templates used for the **Personalization**, **Pedagogy**, and **Satisfaction** metrics are shown in Figures 14–16; SCS is deterministic and uses no judge call.

For the Satisfaction (User Preference) Score, we capture the overall quality of the plan. It answers the question: "Would a typical user be satisfied with this plan as a guide to start their learning journey?" We conduct simulated user preference tests using profile-conditioned LLM judges; the prompt template is shown in Figure 15.

A.6 LLM-as-a-Judge Rubrics

LLM judges are used at *five* places in our data construction and evaluation pipeline (Table 10), but never inside the GRPO training rewards in Section 3.3. Earlier work has shown that vague rubrics make judges brittle to verbosity and persona-keyword bias (Zheng et al., 2023; Liu et al., 2023; Lambert et al., 2024). We therefore specify, for every judge, a Likert anchor scale with *concrete behavioral descriptions* at the 5 / 3 / 1 anchors (intermediate scores 2 and 4 are interpolations), an explicit aggregation rule, and a JSON output schema. SCS is excluded from this inventory because it is computed deterministically from plans and execution logs. Anchors are calibrated against the human-annotation pilot described in Section 5 (Cohen's $\kappa \geq 0.65$).

Table 10: Inventory of LLM-as-a-judge components, where each is used, and the figure that defines its prompt or rubric.

Judge	Where used	Ref.
MAP-PPL execution-effectiveness filter	Data construction, stage 4 (§4)	Fig. 17
Personalization score (Pers.)	Plan-level evaluation (§6.1)	Fig. 18
Plan-level pedagogy (Ped.)	Plan-level evaluation	Fig. 16
Satisfaction score (Sati.)	Post-execution interaction outcome, pairwise	Fig. 19
Dialogue-level pedagogy (PQS)	Post-execution teacher–student traces	Fig. 21

A.7 Anti-Hacking Safeguards

Six protocol-level safeguards prevent LLM-judge-induced reward hacking. (1) The judge model family is disjoint from the family used to synthesize MAP-PPL: plans were generated with Claude Sonnet 4.6, so we evaluate with *Opus* 4.6, GPT-5, and Gemini 3 Pro and never use any of these inside the GRPO reward. (2) Each LLM-judge subdimension is queried in an independent call rather than a multi-criterion prompt. (3) Every pairwise call is run twice with AB and BA orderings; disagreement collapses to Tie. (4) Satisfac-

Table 11: Detailed static-plan ablations on MAP-PPL (305 instances). Pers. and Ped. are normalized to $[0, 1]$ for comparability with TBQ and TS. Rows are grouped by backbone scale.

Variant	TBQ	TS	Pers.	Ped.
8B variants				
PAD-SFT only	0.56	0.81	0.74	0.55
SDP-SFT only	0.51	0.78	0.70	0.51
Joint-align (no GRPO)	0.55	0.78	0.75	0.54
GRPO	**0.69**	**0.82**	0.70	**0.61**
32B variants				
PAD-SFT only	0.64	0.76	0.75	0.56
SDP-SFT only	0.64	0.76	0.69	0.52
Joint-align (no GRPO)	0.69	0.77	0.76	0.55
GRPO	**0.74**	**0.81**	**0.76**	**0.65**

tion is reference-grounded against the gold plan itself. (5) All rule-based and audit metrics (Attempts, Repair Rate, TBQ, TS, SCS, CER) involve no LLM call. (6) We pre-validate the judge configuration on a 100-instance pilot graded by two independent graduate-student annotators (300 annotations per rater on the three Pers. subdimensions) and accept only when Cohen's $\kappa \geq 0.65$ against the human raters, ICC $\geq$ 0.7 across raters, and Krippendorff $\alpha \geq 0.5$ across judge families.

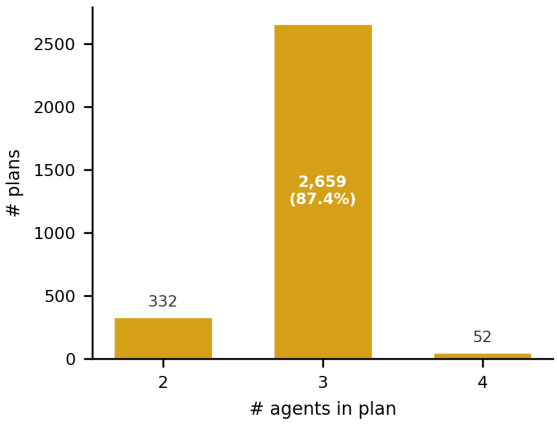
A.8 Component Ablations and Sensitivity

The main ablation analysis is restricted to static plan quality. TBQ and TS cover executable structure, Pers. covers profile grounding, and Ped. covers tutoring-plan pedagogy. Post-execution interaction metrics are omitted here and analyzed separately in the main RQ2 results. The trend matches Figure 4: joint alignment stabilizes the supervised hierarchy, while GRPO supplies the largest gains in tool binding, topology, and pedagogical coverage, especially for the 32B backbone.

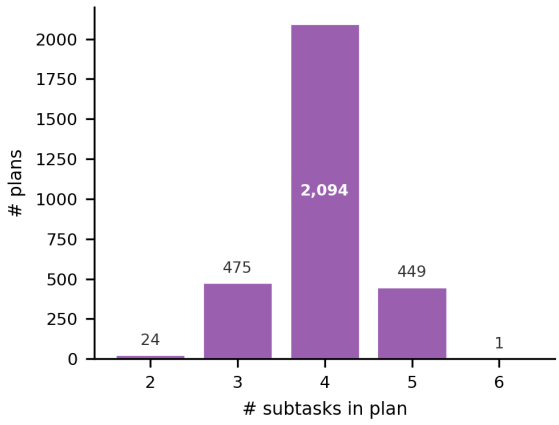
A.9 Case Study

Table 12 expands one representative held-out plan from the plan-generation run: Stack Overflow question `3172100`, *HTML Drag And Drop On Mobile Devices*, paired with profile `1`. The learner reports a mixed background in IT help desk, iOS/Android programming, web design, page layout, marketing, and campaign work, with tags `android`, `java`, `objective-c`, `html`, and `ios`. This instance is useful as a qualitative audit because the target answer is not a single API call: a good plan must explain why desktop drag-and-drop assumptions fail on touch devices, compare several implementation families, and end with a decision procedure the learner can reuse.

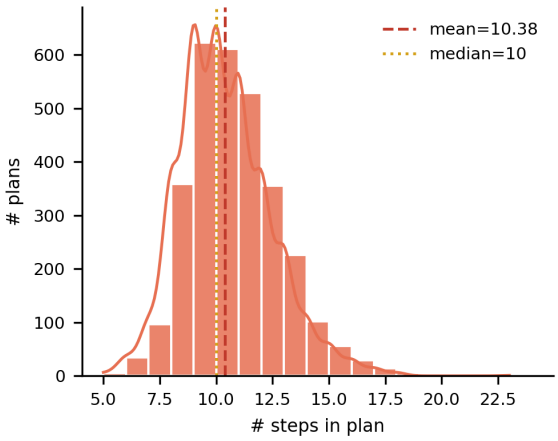
The resulting dependency path is linear where prerequisites matter and parallel where evidence can be gathered independently: `S1-1` $\rightarrow$ `S1-2`; `S2-1`, `S2-2`, and `S2-3` all depend on `S1-1`; `S3-1` consumes the three research briefs; `S3-2` combines the decision matrix with the learner-calibration note; `S4-1` and `S4-2` create complementary prototype artifacts; and `S5-1`–`S5-2` synthesize the lesson and transfer rubric. This example illustrates the intended design pattern: the profile changes the explanatory bridge and artifact choices, the pedagogy progresses from framing to evidence to contrastive examples to self-checks, and the plan remains executable through explicit tools, dependencies, and expected outputs.



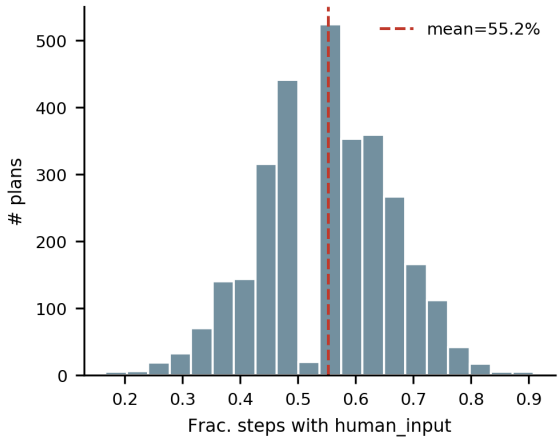
(a) Agents per plan.



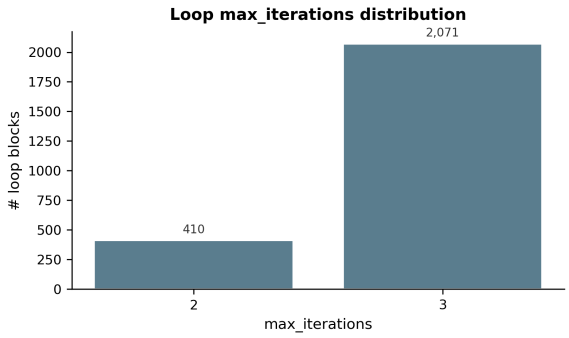
(b) Subtasks per plan.



(c) Steps per plan.

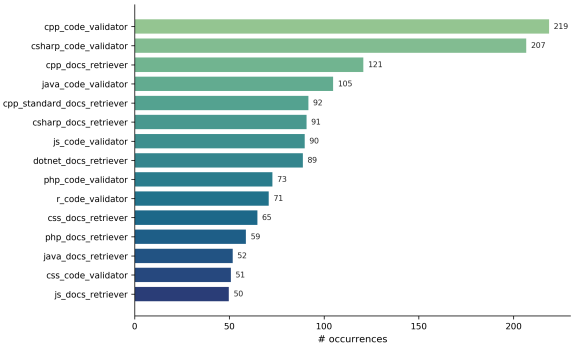


(d) `human_input` step ratio.

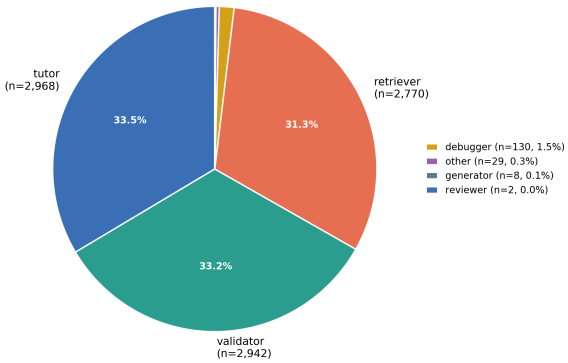


(e) Loop structure (per-plan loop count and max-iteration distribution).

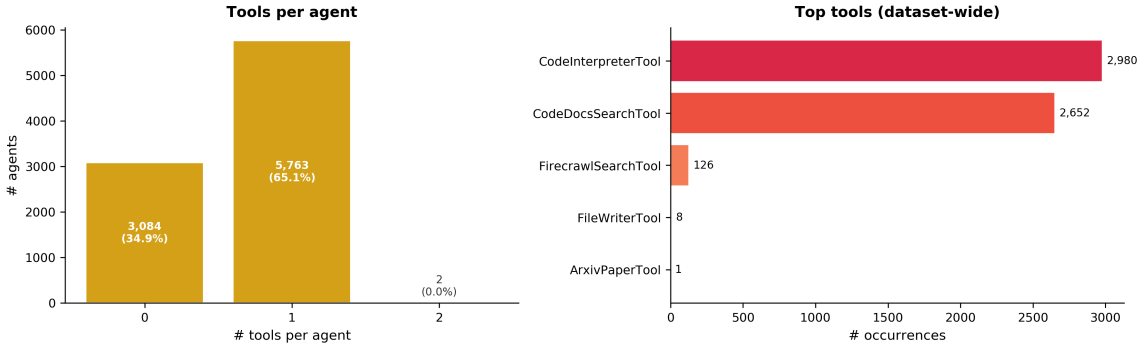
Figure 9: Marginal distributions of plan structural complexity in **MAP-PPL**.



(a) Top-15 agent role names.



(b) Role family share.



(c) Tool counts (left: per-agent; right: dataset-wide).

Figure 10: Agent and tool distributions in **MAP-PPL**.

Table 12: Representative case-study audit for a generated MAP-PPL plan on mobile drag-and-drop. Each row ties an observed plan element to the paper's three qualitative claims: profile grounding, pedagogical scaffolding, and executable multi-agent structure.

Property	**Plan evidence**	**Why this is profile-conditioned**	**Why this is executable / teachable**
Learner calibration	`S1-2` asks the Problem Framer to infer strengths and gaps from the learner profile.	The plan leverages the learner's iOS/Android exposure and basic HTML experience, while avoiding a generic beginner lesson on markup.	The calibration step depends on `S1-1`, so the learner model is formed only after the technical subquestions are extracted.
Agent specialization	The roster contains Problem Framer, Web Platform Researcher, Solution Architect, Prototype Designer, and Teaching Synthesizer.	The Teaching Synthesizer is explicitly tasked with connecting web touch-event concepts to native mobile gesture concepts, matching the profile's mobile-app background.	The roles separate framing, evidence gathering, design comparison, prototype production, and final teaching synthesis rather than mixing all decisions into one monolithic tutor.
Evidence-grounded instruction	`S2-1` uses `CodeDocsSearchTool` for HTML drag-and-drop, touch events, pointer events, and browser gesture behavior; `S2-2` and `S2-3` use `FirecrawlSearchTool` for community workarounds and mobile UX alternatives.	For a learner with practical mobile/web experience, the plan emphasizes platform behavior and compatibility evidence instead of abstract event theory alone.	The plan can be run by an MAS executor because every evidence-gathering step declares a concrete tool and downstream steps depend on the resulting briefs.
Pedagogical sequencing	`S3-1` builds a decision matrix over native HTML5 DnD, jQuery UI touch translation, custom touch/pointer dragging, and mobile-specific fallbacks; `S3-2` orders these from simplest fallback to advanced custom implementation.	The sequence starts from the user's original practical dilemma, then uses mobile-gesture analogies to explain why touch scrolling conflicts with mouse-oriented drag/drop assumptions.	This creates a demonstration-before-application path: first compare solution families, then select a teachable core answer, then create examples.
Runnable artifacts	`S4-1` writes a compact pseudo-implementation for custom touch/pointer dragging; `S4-2` writes a tap-select / tap-drop or long-press fallback pattern.	The artifacts reflect the learner's cross-over context: one path preserves drag behavior, while the other mirrors mobile interaction design where direct drag may be less usable.	Both steps use `FileWriterTool` and feed into `S5-1`, so the final lesson has concrete materials instead of ending at advice.
Transfer check	`S5-2` asks for a mastery checklist covering native HTML5 drag-and-drop, custom touch dragging, scroll interference, and choosing between drag and fallback UX.	The check assesses decision-making across web and mobile contexts, which matches the learner's mixed background more closely than a syntax quiz.	The final assessment depends on the synthesized plan, making transfer a first-class output of the workflow rather than an optional afterthought.

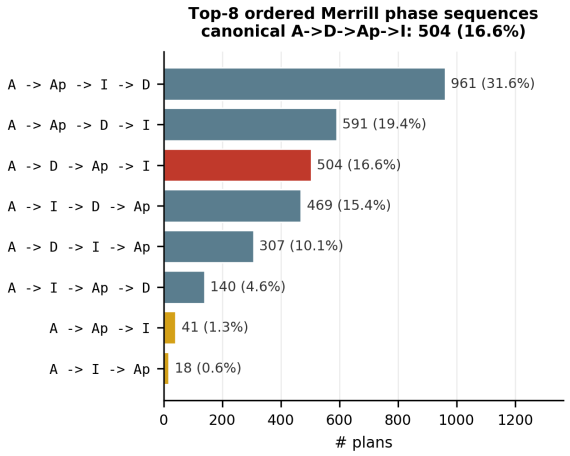
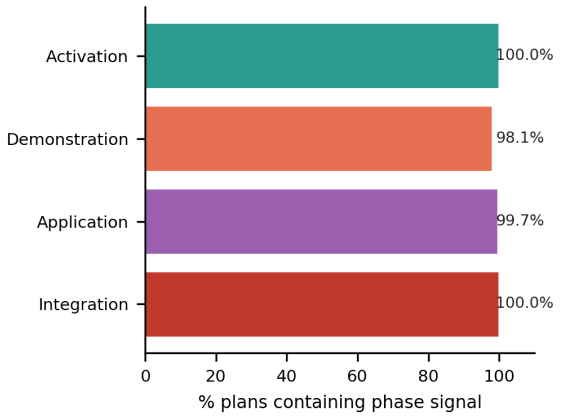
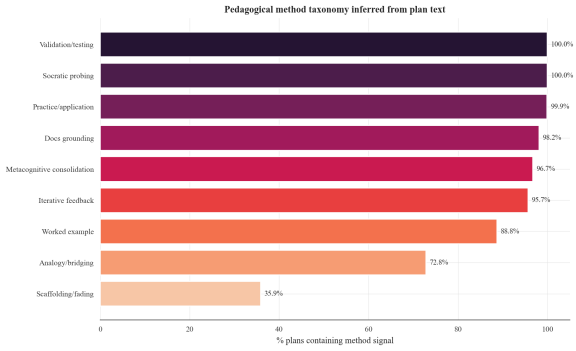


Figure 11: Merrill phase ordering in **MAP-PPL**. The canonical order A→D→Ap→I is one valid pattern but not the dominant template; the most common sequence places learner application and validation before a worked demonstration.



(a) Pedagogical phase coverage (plan / step level).



(b) Coverage of nine canonical instructional methods.

Figure 12: Pedagogical structure of **MAP-PPL** plans: phase coverage at the plan/step level (a) and the coverage of nine canonical instructional methods (b).

Baseline Plan-Generation Prompt (Abridged)

System prompt. Generate a personalized multi-agent plan: a strict JSON specification of agents, subtasks, steps, and execution order. The plan is consumed by a CrewAI-style runtime that instantiates agents and executes steps in order.

Inputs. A StackOverflow question; a learner profile with `about_me` and `top_tags`; and the declared tool pool. Tags indicate topical familiarity rather than mastery.

Tool pool. `FirecrawlSearchTool`, `RagTool`, `CodeInterpreterTool`, `DirectoryReadTool`, `FileReadTool`, `FileWriterTool`, `CodeDocsSearchTool`, `ArxivPaperTool`.

Planner requirements.

- Tailor agent roles, subtasks, explanations, and tool use to the learner and query.
- Produce a plan for teaching the learner to solve the question; do not output the direct answer, a dialogue script, or a lesson transcript.
- Use only tools from the declared pool; `tool` may be `null` when no tool is needed.
- Every step must be scheduled in `execution_order`; dependencies must point to earlier step ids whose outputs are used.
- Output only valid JSON, with no markdown, commentary, extra fields, or trailing commas.

Output schema.

```
{
  "input": {
    "query": "<raw StackOverflow question>",
    "learner": {
      "about_me": "<summary>",
      "top_tags": ["<tag>", "..."]
    }
  },
  "output": {
    "agents": [{
      "agent_role": "<role>",
      "goal": "<goal>",
      "backstory": "<capability>",
      "tools": ["<tool_name>"]
    }],
    "subtasks": [{
      "id": "S1",
      "name": "<name>",
      "subtask_objective": "<objective>",
      "steps": [{
        "id": "S1-1",
        "agent": "<agent_role>",
        "objective": "<objective>",
        "instruction": "<work order>",
        "tool": null,
        "requires_human_input": false,
        "expected_output": "<artifact>",
        "depends_on": []
      }]
    }],
    "execution_order": [
      "S1-1",
      {"loop": {
        "steps": ["S2-1", "S2-2"],
        "condition": "<expr>",
        "max_iterations": 3
      }},
      "S3-1"
    ]
  }
}
```

Figure 13: Abridged shared baseline plan-generation prompt. The full implementation uses this same task definition, tool palette, strict JSON schema, and output constraints; framework-specific wrappers only adapt the prompt to the corresponding API interface.

Personalization Score Prompt

System prompt: You are an expert in computer science education and personalized curriculum design. Your task is to evaluate a generated learning plan based on its *Personalization* for a specific learner.

Context:

- Learner Profile: `{learner_profile_json}`
- Learning Query: `{query_text}`
- Generated Plan: `{generated_plan_json}`

Evaluation instructions: Please rate the plan's Personalization on a scale of 0–1 based on the following criteria. Provide a step-by-step analysis before giving a final score.

1. **Skill & Experience Alignment (0–1):** How well does the plan's starting point, complexity, and choice of technologies match the learner's declared skills and experience?
   - A *high* score means the plan is perfectly pitched (e.g., foundational steps and definitions for a "Beginner"; advanced architecture and optimization for an "Expert").
   - A *low* score means a clear mismatch (e.g., asking a beginner to deploy a Kubernetes cluster).
2. **Goal Orientation (0–1):** How directly does the plan's structure and content contribute to achieving the learner's stated `goal`?
   - A *high* score means every stage and step is relevant and logically moves the learner toward their objective.
   - A *low* score means the plan contains irrelevant steps or fails to address the core of the learning goal.
3. **Background Adaptation (0–1):** Does the plan adapt its examples, agent roles, and explanations to the learner's `background`?
   - A *high* score means the plan shows adaptation (e.g., business-centric examples for a user with a finance background).
   - A *low* score means the plan is generic and ignores the learner's context.

Output format: Return ONLY a single JSON object with the final score and a detailed justification.

```
{
  "personalization_score": <integer from 1 to 10>,
  "justification": "<Your detailed analysis covering all three criteria, explaining the
↪ reasoning behind your score.>"
}
```

Figure 14: Prompt template used to elicit the **Personalization** score from the LLM judge. The judge conditions on the learner profile, query, and generated plan, produces a step-by-step justification across three criteria, and emits a single integer score.

Satisfaction (User Preference) Prompt

System prompt: As a student with profile `{profile}`, which plan would you prefer for learning `{query}`? Consider: skill level matching, engagement, and structural appropriateness.

Candidates:

- Plan A: `{plan_a}`
- Plan B: `{plan_b}`

Output format: Output a single JSON object containing the preferred plan, the judge's confidence, and detailed reasons.

```
{
  "preferred_plan": "A" | "B",
  "confidence": <float in [0,1]>,
  "reasons": "<one or two sentences citing skill-level fit, engagement, and structure>"
}
```

Figure 15: Profile-conditioned pairwise preference prompt used for the **Satisfaction** metric. The judge role-plays the target learner profile and selects between two candidate plans for the same query.

Plan-Level Pedagogy Prompt (Ped.)

System prompt. You are an expert evaluator of multi-agent teaching plans for personalized programming education. Apply the rubric strictly and output valid JSON only. Score concrete step IDs and subtask IDs, not general impressions.

Context.

- Query: `{query_text}`
- Learner profile: `{learner_profile_json}`
- Generated plan: `{generated_plan_json}`
- Accepted answer, used only for the no-direct-answer check: `{accepted_answer}`

Evaluation instructions. We elicit three plan-level pedagogy judgments. **PRR** scores whether the subtask sequence follows progressive teaching: diagnose/probe before explanation, explanation before application, learner attempt before feedback, and no prerequisite concept used before it is introduced. **IAR** scores whether the instructional method adapts to the learner's query-domain expertise: novices should receive worked examples and probe-then-explain scaffolds, experts should receive concise problem-first guidance, and cross-domain learners should receive source-domain analogies. **NDAR** inspects only the first subtask and labels whether it reveals the accepted answer's core solution.

Anchors. For PRR and IAR, use a 1–5 anchored Likert scale: 5 means the target property is consistently observable across the plan, 3 means exactly one major pedagogical mismatch or missing check, and 1 means the plan is pedagogically inverted or generic. For NDAR, output `none`, `partial`, or `full`; `none` means the first subtask probes or scaffolds without revealing the answer, while `full` means it gives away the canonical code/API/algorithm.

Output format.

```
{
  "prr_reasoning": "<=80 words citing step_ids/subtask_ids",
  "prr_score": <int 1-5>,
  "iar_reasoning": "<=80 words citing step_ids/subtask_ids",
  "iar_score": <int 1-5>,
  "ndar_reasoning": "<=80 words comparing the first subtask to the accepted answer",
  "ndar_reveal": "none" | "partial" | "full"
}
```

Figure 16: Prompt template used for the plan-level **Ped.** metric. The implemented judge follows the anchored PRR/IAR/NDAR prompts in the Tier-1 judge script: PRR and IAR are scored on 1–5 scales, NDAR labels first-subtask answer leakage, and rule-based Merrill phase coverage is added separately as SPR.

MAP-PPL Execution-Effectiveness Filter Rubric

Purpose: Admit a synthesized plan into **MAP-PPL** only if it passes the LLM-judged execution-effectiveness gate following the static structure check (Section 4, stage 4).
Inputs to the judge: the Stack Overflow question, the accepted answer, the learner profile I_p, and the candidate plan $\mathcal{P}$.
Scoring dimensions (each on a 1–5 integer Likert scale):

- **D1 — Problem-Solving Progression:** the plan moves from the learner's current state to a complete answer.
- **D2 — Tool-Usage Reasonableness:** every tool call is justified by the step's data needs and the tool pool.
- **D3 — Step Connectivity:** dependencies are semantically sound (each step uses only outputs produced upstream).
- **D4 — Content–Question Alignment:** plan content actually answers the original question and re-uses the accepted answer's core concept.
- **D5 — Personalization Authenticity:** adaptation to I_p is structural, not superficial keyword insertion.

Anchor descriptions (5 / 3 / 1; 2 and 4 are interpolations):
D1 — Problem-Solving Progression.

- **5:** Every stage advances the learner toward the answer; no stage is a detour; final step explicitly produces the artifact the question asks for.
- **3:** Most stages advance the answer but one stage is tangential or redundant; final artifact is reachable but indirectly.
- **1:** The plan terminates without reaching the answer, or includes ≥ 2 stages unrelated to the question.

D2 — Tool-Usage Reasonableness.

- **5:** Every tool comes from the predefined pool, and every assignment matches a concrete I/O need of its step.
- **3:** All tools are in the pool but ≤ 1 step has a loosely justified assignment (e.g., a search tool used where a code-interpreter would suffice).
- **1:** A tool is hallucinated (outside the pool), or ≥ 2 tool assignments are unjustified.

D3 — Step Connectivity.

- **5:** Every dependency edge is justified: the upstream step produces an output the downstream step explicitly consumes.
- **3:** All listed dependencies reference existing steps, but ≤ 1 edge is decorative (no concrete data flow).
- **1:** A dependency references a non-existent step, the graph has a cycle, or ≥ 2 edges have no data flow.

D4 — Content–Question Alignment.

- **5:** The plan teaches the exact concept the accepted answer uses; key terms from the question appear as instructional targets.
- **3:** The plan teaches an adjacent concept that covers most but not all of the answer's solution path.
- **1:** The plan addresses a different problem or copies the accepted answer verbatim instead of teaching it.

D5 — Personalization Authenticity.

- **5:** The plan's *structure* (agent roster, step granularity, prerequisite depth) would change if I_p were replaced by a different profile; profile attributes are referenced concretely.
- **3:** Some surface adaptation (one agent persona, one example domain), but most structure would survive a profile swap.
- **1:** Profile keywords appear in the text but the plan structure is generic; a profile-shuffled version is indistinguishable.

Aggregation and gate: the plan is admitted iff $\min(D_1, \ldots, D_5) \geq 4$ *and* $\sum_i D_i \geq 22$. Otherwise it is returned to stage 3 for regeneration.
Output schema:

```
{
  "D1_progression": <int 1-5>,
  "D2_tool_usage":  <int 1-5>,
  "D3_connectivity": <int 1-5>,
  "D4_alignment":   <int 1-5>,
  "D5_personalization": <int 1-5>,
  "admit": true | false,
  "rationale": "<<=80 words covering every D_i score>"
}
```

Figure 17: Rubric for the MAP-PPL execution-effectiveness filter (stage 4 of dataset construction). Each candidate plan is scored on five 1–5 Likert dimensions; only plans clearing the gate $\min D_i \geq 4$ are admitted, enforcing both executability and structural personalization at data-construction time.

Personalization Score (Pers.) Rubric

Purpose: Score how well a generated plan is tailored to the learner profile I_p. The user-payload prompt template is Figure 14; this rubric is appended to its system prompt.
Inputs: learner profile I_p, query I_q, generated plan $\mathcal{P}'$.
Dimensions (each on a 1–5 integer Likert scale):

- **D1 — Skill & Experience Alignment.**
- **D2 — Goal Orientation.**
- **D3 — Background Adaptation.**

Anchor descriptions:
D1 — Skill & Experience Alignment.

- **5:** Every stage's starting point and tool/library choices are calibrated to the profile's declared proficiency; beginners get explicit setup and definitions, experts get optimization or architecture work without redundant prerequisites.
- **3:** Most stages are calibrated, but one stage either over- or under-shoots (e.g., reintroduces a concept the profile lists as known, or assumes proficiency in a library the profile does not list).
- **1:** ≥ 2 stages misalign with the profile's skill level (e.g., a Kubernetes deployment step for a self-declared "Beginner", or a one-line `for`-loop tutorial for a 5-year Python practitioner).

D2 — Goal Orientation.

- **5:** Every stage and step is traceable to the profile's `goal` field; no stage is purely generic.
- **3:** One stage is partially tangential to the stated `goal` (e.g., a generic "best practices" detour), but most stages remain goal-aligned.
- **1:** The plan's overall trajectory does not match the profile's `goal`, or ≥ 2 stages are off-goal.

D3 — Background Adaptation.

- **5:** Examples, agent personas, and explanations reference the profile's `background` concretely (e.g., portfolio examples for a finance background; CVE-style cases for a security background); adaptation is content-level, not keyword-level.
- **3:** Some examples or agent roles reference the background, but the plan also contains generic examples that ignore it.
- **1:** All examples and agent roles are generic; the `background` field is not actually used in the plan content.

Aggregation: Pers. $= (D_1 + D_2 + D_3)/15 \in [0, 1]$. The judge is also asked to emit an integer 1–10 holistic score `score_1_10` for cross-checking; the official metric is the rubric-averaged value.
Output schema:

```
{
  "D1_skill_alignment":   <int 1-5>,
  "D2_goal_orientation":  <int 1-5>,
  "D3_background_adapt":  <int 1-5>,
  "score_1_10":           <int 1-10>,
  "justification":        "<one paragraph naming each D_i and citing
                           a concrete element of the plan>"
}
```

Figure 18: Likert rubric for the **Personalization** judge. Each of the three sub-criteria is scored on a 1–5 scale with explicit behavioral anchors at 5, 3, and 1; the final Pers. score is the normalized rubric mean.

Satisfaction Score (Sati.) Rubric — Profile-Conditioned Pairwise

Purpose: Estimate user preference rate against the ground-truth plan via profile-conditioned pairwise comparison (the LLM role-plays the target learner). User-payload prompt template: Figure 15.
Inputs: profile I_p, query I_q, two plans `Plan_A` and `Plan_B`, with order randomized; the judge does not know which is the candidate vs. the reference.
Decision dimensions (each is a ternary vote `A/B/Tie`):

- **V1 — Skill-Level Fit.**
- **V2 — Engagement.**
- **V3 — Structural Appropriateness.**
- **$V_\star$ — Overall preference.**

Anchor descriptions (X stands for the plan being judged better on that dimension):
V1 — Skill-Level Fit.

- **X wins:** Plan X's entry point and explanation depth match the profile's proficiency at ≥ 1 stage where the other plan over- or under-shoots.
- **Tie:** Both plans calibrate to the same level; differences are stylistic.

V2 — Engagement.

- **X wins:** Plan X ties tasks to the profile's stated `goal` or `background` (concrete projects/examples the learner would find motivating).
- **Tie:** Both plans are equally generic, or equally well personalized.

V3 — Structural Appropriateness.

- **X wins:** Plan X's subtask granularity and dependency density are closer to what a learner with this profile could realistically execute in one session.
- **Tie:** Granularity is comparable.

$V_\star$ — Overall: The judge re-reads both plans and casts a holistic preference. If $V_\star$ disagrees with the majority of V_1, V_2, V_3, the judge must justify the deviation.
Anti-bias safeguards (enforced in the system prompt): (i) order is randomized with probability 0.5; the judge does not know which plan is the reference; (ii) the judge is instructed to ignore plan length, formatting, and persona-keyword density; (iii) ties are allowed and not penalized.
Aggregation: per pairwise call, $\text{Sati}_{\text{call}} = \phi(V_\star)$ with $\phi(\text{A}) = 1, \phi(\text{Tie}) = 0.5, \phi(\text{B}) = 0$ (relative to the candidate plan after order de-randomization). The reported metric is the mean over the test set.
Output schema:

```
{
  "V1_skill_fit":     "A" | "B" | "Tie",
  "V2_engagement":    "A" | "B" | "Tie",
  "V3_structure":     "A" | "B" | "Tie",
  "Vstar_overall":    "A" | "B" | "Tie",
  "confidence":       <float in [0,1]>,
  "reasons":          "<<=40 words; must cite at least one
                        concrete subtask from each plan>"
}
```

Figure 19: Profile-conditioned pairwise rubric for the **Satisfaction** judge. The judge votes on three persona-grounded dimensions plus an overall preference; explicit anti-bias safeguards (order randomization, length-/keyword-ignore instruction, tie allowance) follow the protocol of Zheng et al. (2023).

MAS Simulation Outcome Rubric (Post-execution)

Purpose: Score the execution trace produced when a generated plan is handed to a multi-agent runtime (GPT-4o teacher agent + GPT-4o-mini learner agent under a topic-blindness mechanism) on the online interaction subset (Section 5). The judge inspects the trace, not the original plan.
Inputs: learner profile I_p, query I_q, generated plan $\mathcal{P}'$, and the full simulation transcript $\mathcal{H}$ (agent turns, tool calls, learner replies).
Dimensions (each on a 1–5 integer Likert scale):

- **S1 — Plan Executability.** What fraction of plan steps actually ran in $\mathcal{H}$?
- **S2 — Tool-Call Validity.** Were tool invocations syntactically and semantically correct?
- **S3 — Inter-Agent Coordination.** Did agents hand off cleanly without dead-locks, duplicated work, or contradictory instructions?
- **S4 — Topic-Blind Learner Coverage.** Did the teacher elicit understanding from a learner who initially lacks the topic vocabulary?
- **S5 — Goal Achievement.** Did the trace end with the deliverable the query asks for?

Anchor descriptions:
S1 — Plan Executability. **5:** every step in $\mathcal{P}'$ has a matching action in $\mathcal{H}$ that completed without an unhandled error. **3:** $\geq 70\%$ of steps executed; failures are recoverable (retry, alternate tool). **1:** $\leq 40\%$ of steps executed, or the trace aborted before the midpoint.
S2 — Tool-Call Validity. **5:** all tool calls use pool-valid signatures and consume only outputs previously produced. **3:** one tool call is malformed but recovered. **1:** ≥ 2 tool calls hallucinate arguments, or a tool outside the pool is invoked.
S3 — Inter-Agent Coordination. **5:** each agent's turn cites a prior turn it depends on; no contradictory instructions; no dead-locks. **3:** one minor coordination glitch (duplicated subtask), resolved within two turns. **1:** agents repeatedly contradict each other, or the trace dead-locks before goal completion.
S4 — Topic-Blind Learner Coverage. **5:** the teacher detects the learner's vocabulary gaps and introduces concepts before using them; the learner's confusion is resolved by the next teacher turn. **3:** the teacher introduces most concepts on demand but ≥ 1 confusion goes unaddressed. **1:** the teacher consistently assumes vocabulary the learner does not have.
S5 — Goal Achievement. **5:** the trace ends with a working artifact that solves the query, validated by the learner agent. **3:** a partial artifact is produced but at least one acceptance criterion is unmet. **1:** no artifact is produced or the produced artifact does not address the query.
Aggregation: $\text{MAS}_{\text{sim}} = (S_1 + \cdots + S_5)/25 \in [0, 1]$; gate: a run *counts* as a successful execution iff $S_1 \geq 4$ and $S_5 \geq 3$.
Output schema:

```
{
  "S1_executability":  <int 1-5>,
  "S2_tool_validity":  <int 1-5>,
  "S3_coordination":   <int 1-5>,
  "S4_blind_coverage": <int 1-5>,
  "S5_goal":           <int 1-5>,
  "executed_steps_pct": <float in [0,100]>,
  "rationale": "<<=100 words; cite turn IDs from the transcript>"
}
```

Figure 20: Rubric for the post-execution **MAS simulation outcome** judge. The judge consumes the full execution transcript $\mathcal{H}$ (not the static plan) and verifies that profile-conditioned plans are actually runnable under a realistic topic-blind learner.

Dialogue-Level Pedagogy Rubric (Post-execution)

Purpose: Score the pedagogical quality of the teacher–learner dialogue produced inside a MAS simulation (Section 5). Unlike the plan-level pedagogy reward used during GRPO training, this rubric scores *realized teaching behavior* in the trace.
Inputs: learner profile I_p, query I_q, dialogue transcript $\mathcal{H}_{\text{dial}}$ (teacher and learner turns only; tool-call turns are stripped).
Dimensions (each on a 1–5 integer Likert scale):

- **P1 — Pedagogical Scaffolding.** Are concepts introduced via explicit prerequisite turns before use?
- **P2 — Adaptivity.** Does the teacher detect and respond to learner confusion?
- **P3 — Conceptual Accuracy.** Are explanations technically correct (no hallucinated APIs/behaviors)?
- **P4 — Profile Appropriateness.** Does the teaching register match I_p's skill and background?
- **P5 — Goal Transfer.** By the final turn, does the learner demonstrate (in their own words or code) that they can solve the original I_q?

Anchor descriptions:
P1 — Pedagogical Scaffolding. **5:** every new concept is introduced by a dedicated teacher turn before being applied; the dialogue contains explicit "before we do X, recall Y" moves. **3:** most concepts are scaffolded but one is used before being introduced; the learner asks for clarification. **1:** ≥ 2 concepts are used cold; the learner is left to infer prerequisites.
P2 — Adaptivity. **5:** every learner-confusion turn ("I don't understand X") is followed by a teacher turn that targets exactly that confusion with a concrete example or rephrasing. **3:** most confusions are handled, but one is answered with a repeat of the original explanation. **1:** the teacher proceeds along the plan even after the learner has stated confusion.
P3 — Conceptual Accuracy. **5:** all teacher claims are technically correct (verified against the accepted answer); code snippets compile; APIs exist. **3:** one minor inaccuracy (e.g., a deprecated flag) that does not affect the final solution. **1:** ≥ 2 technical errors, or a load-bearing claim is wrong.
P4 — Profile Appropriateness. **5:** register, vocabulary, and example domains visibly match I_p; an outside observer could correctly guess the profile from $\mathcal{H}_{\text{dial}}$ alone. **3:** register is broadly appropriate but slips into generic language at ≤ 1 stretch. **1:** the dialogue would be indistinguishable across very different profiles.
P5 — Goal Transfer. **5:** the learner's final turn solves a held-out instance of I_q correctly with no further teacher hint. **3:** the learner solves it with one hint or one partial mistake. **1:** the learner fails the held-out check, or only repeats the teacher's last turn.
Aggregation: $\text{Ped}_{\text{dial}} = (P_1 + \cdots + P_5)/25 \in [0, 1]$. Pairwise variant: pass two transcripts to the judge with order randomization and ask for per-dimension A / B / Tie votes plus an overall verdict; this variant is used when reporting head-to-head against the strongest baseline.
Output schema:

```
{
  "P1_scaffolding": <int 1-5>,
  "P2_adaptivity":  <int 1-5>,
  "P3_accuracy":    <int 1-5>,
  "P4_profile_fit": <int 1-5>,
  "P5_goal_transfer": <int 1-5>,
  "held_out_check_passed": true | false,
  "rationale": "<<=120 words; cite turn IDs and the held-out
                 check outcome>"
}
```

Figure 21: Rubric for the post-execution **dialogue-level pedagogy** judge. Unlike the plan-level pedagogy reward used during GRPO training, this rubric scores *realized teaching behavior* in the teacher–learner transcript, including a held-out goal-transfer check that the learner must pass without further hints.