Bridging ClarifyCoder to Production: A Methods Blueprint

Abstract

We present a methods-focused blueprint for deploying ClarifyCoder—a clarification-aware code LLM—in real developer workflows. We formalize tasks, training data, modeling, orchestration, deployment in IDE and GitHub settings, education use-cases, evaluation metrics with statistical tests, ablations, and security/compliance requirements. Our design operationalizes ClarifyCoder's clarify-then-code paradigm [1] for repository-scale issues (e.g., SWE-bench/SWE-bench+) [2, 3], execution-grounded generalization (LiveCodeBench) [4], and repository completion (RepoBench) [5], while mitigating package/API hallucinations [7] and aligning with field evidence on coding assistants [6] and public-sector trials [8].

1 Problem Framing and Desiderata

ClarifyCoder improves alignment by asking a targeted question when the specification is ambiguous and only then producing code [1]. We aim to convert this capability into reliable *system performance* in real contexts: (i) detect underspecification; (ii) ask one minimally-sufficient clarifying question; (iii) produce reviewable, runnable changes integrated into IDEs, Issues/PRs/CI; (iv) ensure auditability and safety.

2 Tasks and Data

T1: Issue Triage with Clarification. Input: GitHub Issue text + minimal repo context; Output: `ASK` (one question) or `PLAN/CODE` (diff/tests). Evaluate on SWE-bench and recent SWE-bench+ issues [2, 3].
T2: Patch-Sized Editing with Ambiguity Detection. Input: IDE selection/diff hunk + local failing tests. Output: `ASK` or `PATCH` (+ test). Evaluate on LiveCodeBench execution tracks and repository-level tasks [4, 5].
T3: Pedagogical Programming with Rubric Grounding. Input: assignment prompt + rubric (clarifiable vs. non-clarifiable). Output: `ASK` → `CODE` + rationale mapping assumptions to rubric. Evaluate via LMS logs and controlled studies.

Ambiguity-Enriched Training. Start with ClarifyCoder's synthetic ambiguity + instruction tuning recipe [1]. Extend with *hard negatives* mined from SWE-bench(+) issues that are not resolvable without environment specs (OS, Python version, repro steps), labeling missing fields [2, 3]. Build retrieval corpora with repository graphs (imports, tests, build metadata) to make dependency constraints explicit, reducing package/API hallucination risks [7].

3 Modeling and Orchestration

Two-Head Controller (ASK vs. ACT). Add a small policy head atop the base to choose `ASK` or `ACT`. Train with Hamming loss against oracle labels. Labels derive from (a) rubric of required fields (Issues/PRs), (b) flakiness/trace analysis, (c) low retrieval confidence [1, 2].

Ledger-Conditioned Generation. Represent clarifications as a key–value *ledger* appended to the prompt. Require strict schema:

- `ASK: <one question>`, or
- `CODE:` fenced `diff/python` + `EXPLAIN:` (2–5 lines citing ledger keys used).

Retrieval and Constraints Layer. Before generation, run scoped retrieval over the repo (ripgrep/LSP index) to fetch files referenced by traces, imports, and implicated tests. Enforce a *dependency allowlist* from `requirements.*`/lockfiles and mask non-approved package tokens in decoding to mitigate package hallucination/slopsquatting [7].

Self-Checking and Test-First Proposals. When acting, generate tests first where missing, then the minimal patch that makes tests pass in a sandbox. Prefer execution-grounded evaluation (LiveCodeBench style) [4].

4 Deployment

IDE Extension. On save/selection, compute an ambiguity score (policy head). If > threshold, show a non-modal banner: "Ambiguity detected—1 question." Present `ASK` in a side panel; the user's answer updates the ledger; re-invoke the model to produce a *scratch diff* + tests. Controls: tests-first toggle, security scan before apply, max LOC changed. Field evidence shows productivity gains but significant human editing [6, 8].

GitHub Bot. For Issues, on `opened/edited` events, triage; if `ASK`, post one question + label `needs-info`; else post acceptance criteria and a minimal repro. For PRs/CI, attach failing trace summaries; if ambiguous, `ASK`; else propose a suggested change bounded by K LOC [2, 3].

Education (LMS). Instructors supply a YAML rubric of clarifiable fields. Student flow caps at ≤ 2 questions before code; return solution plus an "assumption map." This scaffolds specification skills and yields auditable ledgers for grading.

5 Evaluation Protocol

Ambiguity Detection. AUC-PR for `ASK` vs. `ACT`. Expected Clarification Count (ECC) ≈ 1.0 [1].
End-to-End Resolution. SWE-bench(+) success@K [2, 3], RepoBench completion [5], LiveCodeBench pass rate [4].
Human-in-the-Loop UX. Time-to-first-commit, diff churn, edit ratio (human edits / assistant lines) [6, 8].
Safety. Package Hallucination Rate; Slopsquat Vulnerability Rate [7].
Education. Rubric alignment (Cohen's κ) and learning gain.

Statistical Testing. Clustered A/B tests by repo/class. Mixed-effects models with random intercepts. Wilson CIs and McNemar's test for paired SWE-bench tasks [2, 3].

6 Ablations

(1) Clarify policy: `ASK@1` vs. `ASK@N` vs. none [1]. (2) Retrieval scope [2]. (3) Dependency guard [7]. (4) Test-first vs. patch-first [4]. (5) Base/fine-tuning controls [1].

7 Security and Compliance

Send only retrieved snippets; attach ledger + retrieval IDs; sandbox execution; mirror registries with signature checks [7].

8 Expected Outcomes and Failure Modes

We expect reduced time-to-first-commit, fewer `needs-info` loops, higher SWE-bench(+) success@K, and better rubric alignment [2, 3, 6]. Failure modes: over-questioning, API hallucinations, classroom over-reliance.

9 Reproducibility

Release prompts, policy-head weights, retrieval rules, and a Docker evaluation harness for SWE-bench(+), RepoBench, and LiveCodeBench [4].

References

[1] J. Wu et al. *ClarifyCoder: Clarification-Aware Code LLMs*. arXiv preprint arXiv:2504.16331, 2025.

[2] C. Jimenez et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* arXiv preprint arXiv:2310.06770, 2023.

[3] C. Jimenez et al. *SWE-bench+: Scaling Realistic Software Engineering Benchmarks*. arXiv preprint arXiv:2410.06992, 2024.

[4] N. Muennighoff et al. *LiveCodeBench: Evaluating Code Generation in the Wild*. arXiv preprint arXiv:2403.07974, 2024.

[5] Y. Zhang et al. *RepoBench: Benchmarking Repository-Level Code Generation*. arXiv preprint arXiv:2306.03091, 2023.

[6] C. Bird et al. *Large-Scale Field Study of AI Coding Assistants in Practice*. arXiv preprint arXiv:2406.17910, 2024.

[7] A. Narayanan et al. *Mitigating Package Hallucination and Slopsquatting Risks in Code LLMs*. arXiv preprint arXiv:2406.10279, 2024.

[8] IT Pro. *UK government AI coding assistant trial reports developer time savings*. News article, 2024. https://www.itpro.com/software/development/uk-government-ai-coding-assistant-trial-developer-time-savings