RepoDesign: Codebase-Aware Multimodal System Design Planning

Andrea Jimenez Fernandez
MIT
andrejim@mit.edu

Cerine Hamida
MIT
chamida@mit.edu

Kevin Power
MIT
kevpower@mit.edu

Abstract

Large language models can discuss system design fluently, yet their advice is fundamentally ungrounded: they suggest adding a Redis caching layer without checking whether the codebase already has one, or recommend Kubernetes for a 200-user internal tool. We introduce **RepoDesign**, a multimodal agent pipeline that accepts a product specification, an existing code repository, and (optionally) architecture diagrams, and produces a codebase-aware implementation plan with real file paths, scale-appropriate technology choices, and executable ticket descriptions. Our core contributions are: (1) a structured **Repo Intermediate Representation (Repo IR)** capturing repository architecture through deterministic code analysis augmented by LLM summarization; (2) a **multimodal fusion mechanism** grounding visual architecture diagrams against the Repo IR; and (3) a **scale-aware design reasoning framework** conditioning recommendations on project constraints (team size, user count, budget) rather than defaulting to enterprise-grade patterns. We fine-tune Qwen3-VL-235B-A22B-Instruct via full LoRA (via Tinker) with SFT and GRPO, evaluated using a novel **Repo Grounding Score** measuring whether generated plans reference files and modules that actually exist in the target repository.

1 Introduction

System design is one of the most consequential activities in software engineering; a poor architectural decision in week one can cost months of rework. Yet today's AI tools leave a critical gap. Code-generation tools (GitHub Copilot, Cursor) excel at implementing functions but offer no guidance on *what to build or where to put it* within an existing codebase. Conversational assistants (ChatGPT, Claude) discuss system design eloquently but their recommendations are generic, decoupled from any repository, blind to existing infrastructure, and biased toward over-engineered solutions. As Kleppmann [10] emphasizes, the value of a design checklist is *consciously deciding which items matter for your specific system*. Current AI tools fail at this: they cannot see you already use Django with PostgreSQL, miss that `docker-compose.yml` reveals a single-server deployment, and recommend sharding regardless of row count.

Broad Impact. A repo-grounded, scale-appropriate design tool would democratize architectural expertise, giving junior engineers guidance calibrated to their actual context rather than Silicon Valley-scale assumptions, reducing architectural debt [2].

Intellectual Merit. Faithfully representing repository architecture requires fusing heterogeneous signals (dependency graphs, ORM models, API decorators, infrastructure configs, visual diagrams). Generating grounded plans avoiding hallucinated paths requires structured reasoning current LLMs struggle with. Scale-aware reasoning demands learning *when not to recommend complexity*, counter to training data biases, and the multimodal aspect requires non-trivial cross-modal alignment.

2 Related Work

2.1 Code-Aware Language Models

CodeBERT [5] and CodeT5 [19] introduced code pre-training; StarCoder [12] and Code Llama [16] scaled to long-context generation. RepoCoder [23], SWE-Agent [22], and SWE-bench [8] pushed toward repo-level reasoning. OpenHands [18] and Devin [4] extend to autonomous multi-step engineering, while Qwen2.5-Coder [7] and DeepSeek-Coder-V2 [24] show open-weight models approach frontier performance. *Why build a Repo IR when models can ingest long contexts?* LongBench v2 shows persistent failures in far-context utilization: attention diffusion, structural underuse, and high inference cost. **Repo IR is a structural scaffold**: it deterministically surfaces high-signal structure (routes, ORM topology, infra configs) into a stable representation planners reliably attend to. Despite this progress, existing systems focus on code generation and bug fixing rather than *architectural reasoning*.

2.2 Automated Software Architecture and Agentic Workflows

Early work used formal ADLs [14]; more recently, Chen et al. [3] explored LLM-based UML generation, while ChatDev [15] and MetaGPT [6] simulate multi-agent teams producing designs and code. However, these generate architectures *de novo* without codebase awareness and target only greenfield projects. Flow-engineering approaches optimize for patch correctness (compilation, tests) rather than *design correctness* (grounded in existing architecture, appropriate for constraints). RepoDesign's "verifier" is the repository itself (via Repo Grounding Score) and a constraint rubric (via Scale Appropriateness).

2.3 Multimodal Diagram Understanding and Grounding

Qwen2-VL [20] and LLaVA [13] interpret diagrams and screenshots; prior work explores UML understanding [9] and whiteboard-to-structure translation. No prior work *grounds* visual architecture diagrams against an actual codebase, e.g., mapping "User Service" in a diagram to `src/services/user/`. Related GUI grounding work maps UI elements to DOM/accessibility trees. RepoDesign's diagram grounding is the architectural analogue: **visual-to-structured alignment enabling downstream action**.

2.4 Constraint-Aware Planning and Preference Alignment

RAG [11] and tool-augmented LLMs [17] address factual grounding; SayCan [1] and Plan-and-Solve [21] explore LLM planning. For system design, correctness requires *appropriateness under constraints*. Recent preference optimization enforces constraint satisfaction beyond outcome correctness. RepoDesign applies this via GRPO over contrastive "scale-tier" pairs, penalizing both over- and under-engineering. **Gaps:** prior work (a) generates code without architectural reasoning, (b) reasons about architecture without codebase awareness, (c) understands diagrams without code grounding, or (d) ignores scale. RepoDesign unifies all four.

3 Research Ideas

3.1 Repo IR Extraction and Grounding (Andrea Jimenez Fernandez)

Idea: Build a structured Repo IR combining deterministic static analysis (dependency graphs via pipreqs/madge, API route extraction via AST parsing, ORM and infrastructure config parsing) with LLM summarization. Core question: does a fine-tuned model produce faithful Repo IRs that significantly reduce hallucinated file paths in downstream plans? **Novelty:** No prior work defines a canonical structured repository-architecture representation for downstream design tasks. The Repo IR schema is itself a reusable benchmark artifact.

3.2 Multimodal Architecture Diagram Fusion (Cerine Hamida)

Idea: Accept architecture diagrams (PlantUML, Mermaid, whiteboard photos, cloud screenshots) and align diagram components with Repo IR entities, e.g., mapping "Auth Service" to `src/auth/`

or an "API Gateway $\rightarrow$ Database" edge to the actual call chain. **Novelty:** We propose *diagram grounding*: cross-modal alignment between visual architecture and code structure, distinct from diagram generation or isolated understanding.

3.3 Scale-Aware Design Reasoning (Kevin Power)

Idea: Condition recommendations on scale constraints (user count, team size, budget) via GRPO over contrastive pairs where "preferred" plans match the tier and "rejected" plans over- or under-engineer. We use a 20-dimension framework from Kleppmann [10] (performance, consistency, caching, replication, partitioning, security, failure resilience) across four tiers: *hobby* (<1k users, 1 dev), *startup* (1k–50k, 2–5 devs), *growth* (50k–1M, 10–30 devs), *enterprise* (1M+, 30+ devs). For example, at hobby tier caching is "none, your DB is fast enough"; at growth, "Redis with cache-aside"; at enterprise, multi-layer caching with CDN and stampede prevention. This framework serves as both training signal and evaluation rubric. **Novelty:** No prior work trains or evaluates LLMs on scale-appropriate system design.

4 Planned Execution

4.1 Datasets

Repo IR: 5,000–20,000 open-source GitHub repos across scales ($\sim$ 2,500 small, $\sim$ 1,500 medium, $\sim$ 750 growth, $\sim$ 250 large/enterprise); silver-label Repo IR JSON via Deepseek Reasoner; 50–100 manually corrected gold examples. **Implementation Plans:** 2–3 synthetic PRDs per repo plus negative examples (hallucinated paths, scale-mismatched plans) for GRPO pairs. **Diagrams:** $\sim$100–150 diagram-repo pairs from `.puml`/`.mmd` files, supplemented with synthetically degraded renders. **Scale Pairs:** 50+ contrastive scenario pairs using the 20-dimension framework as ground truth.

4.2 Evaluation Metrics

Repo Grounding Score (RGS): % of file paths in the plan that exist in the repo; fully deterministic, no LLM-as-judge. **Plan Completeness:** Scale-conditioned checklist across 20 design dimensions (a 500-user plan is not penalized for omitting sharding; a 5M-user plan is). **Scale Appropriateness:** Whether recommended technologies match the stated tier. **Constraint Satisfaction:** % of hard spec constraints respected. **Format Compliance:** Valid JSON conforming to defined schemas. **Human Evaluation:** 10–20 engineers compare our model vs. GPT-4: (a) "execute Monday morning," (b) scale match, (c) code accuracy.

4.3 Experimental Setup

RQ1 (Grounding): Repo IR vs. naive RAG. *H:* $\geq$30% higher RGS. **RQ2 (Multimodal):** Diagrams vs. text-only. *H:* Higher completeness, fewer hallucinations. **RQ3 (Scale):** GRPO vs. SFT-only. *H:* $\geq 2\times$ more scale-appropriate recommendations. **RQ4 (End-to-End):** RepoDesign vs. GPT-4. *H:* Preferred $\geq$60% on "execute Monday morning."

Baselines: (1) GPT-4 with system design prompt; (2) GPT-4 with naive RAG; (3) SFT-only (no GRPO). **Training:** Qwen3-VL-235B-A22B-Instruct, full LoRA via Tinker. Stage 1: SFT on Repo$\rightarrow$IR and Spec+IR$\rightarrow$Plan. Stage 2: GRPO on scale-contrastive preference pairs.

Risks: (1) Noisy IR for poorly structured repos (mitigated by stratified evaluation). (2) Insufficient data (mitigated by augmentation and RAG fallback). (3) Scarce diagram data (mitigated by synthetic renders; evaluated as delta over text-only). (4) Small human study (mitigated by paired comparisons with confidence intervals).

5 Task Division and Timeline

Andrea (Repo IR & Data): Deterministic analysis tools, IR schema, scraping pipeline, RGS metric. **Cerine (Multimodal & Training):** Diagram extraction, multimodal formatting, SFT/GRPO runs, model evaluation. **Kevin (Scale & Evaluation):** Scale rubric, contrastive pairs, human evaluation, Spec Normalizer, demo UI. All members co-write the final report with cross-functional code reviews. GitHub: `https://github.com/andreajf94/Multimodal`.

Weeks	**Milestone**
1–2	Define JSON schemas (Spec, IR, Plan). Build IR extraction v1. Curate repo list. Build Spec Normalizer. Set up Tinker/LoRA.
3–4	Run IR extraction. Generate silver-label IRs. Validate 50+. Generate PRDs, gold plans, contrastive pairs. Collect diagrams.
5–6	SFT runs (Repo$\rightarrow$IR; Spec+IR$\rightarrow$Plan). Implement automated metrics. Evaluate and iterate.
7–8	Build GRPO dataset. Run GRPO. Compare SFT vs. GRPO. Build demo UI. Begin human eval recruitment.
9–10	Full evaluation across baselines. Human eval (10–20 participants). Draft final report.
11–12	Final report. Live demo. Submit code, data, model to GitHub.

References

[1] M. Ahn et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv:2204.01691*, 2022.

[2] T. Besker, A. Martini, and J. Bosch. Managing architectural technical debt: A unified model and systematic literature review. *Journal of Systems and Software*, 135:1–16, 2018.

[3] M. Chen et al. Towards automated UML generation from natural language requirements using LLMs. *ASE Workshop*, 2023.

[4] Cognition Labs. Devin: The First AI Software Engineer. Blog post, 2024.

[5] Z. Feng et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *EMNLP*, 2020.

[6] S. Hong et al. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. *ICLR*, 2024.

[7] B. Hui et al. Qwen2.5-Coder Technical Report. *arXiv:2409.12186*, 2024.

[8] C. E. Jimenez et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ICLR*, 2024.

[9] T. Kimura et al. Understanding UML diagrams with vision-language models. *ICSE Companion*, 2023.

[10] M. Kleppmann. *Designing Data-Intensive Applications*. O'Reilly, 2017.

[11] P. Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *NeurIPS*, 2020.

[12] R. Li et al. StarCoder: May the Source Be With You. *arXiv:2305.06161*, 2023.

[13] H. Liu et al. Visual Instruction Tuning. *NeurIPS*, 2023.

[14] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE TSE*, 26(1):70–93, 2000.

[15] C. Qian et al. ChatDev: Communicative Agents for Software Development. *ACL*, 2024.

[16] B. Rozière et al. Code Llama: Open Foundation Models for Code. *arXiv:2308.12950*, 2023.

[17] T. Schick et al. Toolformer: Language Models Can Teach Themselves to Use Tools. *NeurIPS*, 2024.

[18] X. Wang et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *arXiv:2407.16741*, 2024.

[19] Y. Wang et al. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. *EMNLP*, 2021.

[20] P. Wang et al. Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. *arXiv:2409.12191*, 2024.

[21] L. Wang et al. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. *ACL*, 2023.

[22] J. Yang et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *arXiv:2405.15793*, 2024.

[23] F. Zhang et al. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. *EMNLP*, 2023.

[24] Q. Zhu et al. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. *arXiv:2406.11931*, 2024.