BACKENDFORGE: Benchmarking Agentic End-to-End Code Generation with Backend Services

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly used in agentic coding settings, where they can inspect files, execute commands, run tests, observe failures, and iteratively revise code. This shift raises a central evaluation question: can an agentic LLM generate an end-to-end software artifact that is both deployable and behaviorally correct under execution? Backend services provide a controlled but realistic substrate for this evaluation. Their APIs expose application-level executable semantics, and deployed behavior can be checked deterministically against an OpenAPI contract through black-box HTTP interactions. We introduce BACKENDFORGE, a benchmark of 56 contract-defined backend generation tasks rewritten from real open-source applications. Given a visible specification and an OpenAPI contract, an LLM must generate a Dockerized service that is built, deployed, and evaluated only through HTTP tests. To strengthen evaluation without introducing hidden requirements, BACKENDFORGE uses a test agent and a code agent to co-evolve the test oracle and reference service, where the test agent proposes specification-grounded backend tests and the code agent repairs the reference implementation. Although the best-performing model, GPT-5.5, succeeds on 55.4% of tasks under the base oracle, it succeeds on only 28.6% under the final oracle. This gap suggests that current LLMs can implement many local API behaviors, but still struggle to produce complete backend services.

1 Introduction

Large language models (LLMs) are rapidly changing the unit of software generation. Early code generation systems primarily targeted localized assistance, such as completing a function, solving a programming problem, or producing a small code snippet (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021). Subsequent systems and benchmarks moved toward repository-level development, where models modify existing codebases to resolve bugs or implement requested changes (Jimenez et al., 2024). More recently, LLMs are used in agentic coding settings where they can inspect project files, execute commands, run tests, observe failures, and iteratively revise code within a workspace (Yang et al., 2024; Anthropic, 2026a; OpenCode, 2026). As LLM coding workflows become more interactive and autonomous, user expectations are shifting from receiving isolated code fragments toward obtaining usable software artifacts (Yao et al., 2023; Wu et al., 2024; Lu et al., 2021; Hong et al., 2024). For such artifacts, usefulness depends not only on whether code compiles or local tests pass, but on whether the generated system can be built, deployed, and interacted with through its intended interfaces.

Evaluating end-to-end software generation requires balancing realism against deterministic evaluation (Zhou et al., 2024; Xie et al., 2024). A fully open-ended benchmark could ask models to infer requirements from underspecified user requests, design an interface, implement the system, and deploy it (Qian et al., 2024; Chen et al., 2024). Such a setting would be realistic, but difficult to evaluate deterministically because failures could arise from requirement interpretation, interface design, implementation, or deployment (Ezzini et al., 2021; Kapoor et al., 2024). In BACKENDFORGE, we instead isolate the contract-realization problem: given an explicit natural-language specification and an OpenAPI contract, can an agentic LLM construct a deployable backend service whose externally observable behavior satisfies that contract?

Existing benchmarks evaluate important components of agentic coding, but they do not directly evaluate whether an LLM can generate a deployable software service that behaves correctly after deployment. Repository repair settings such as

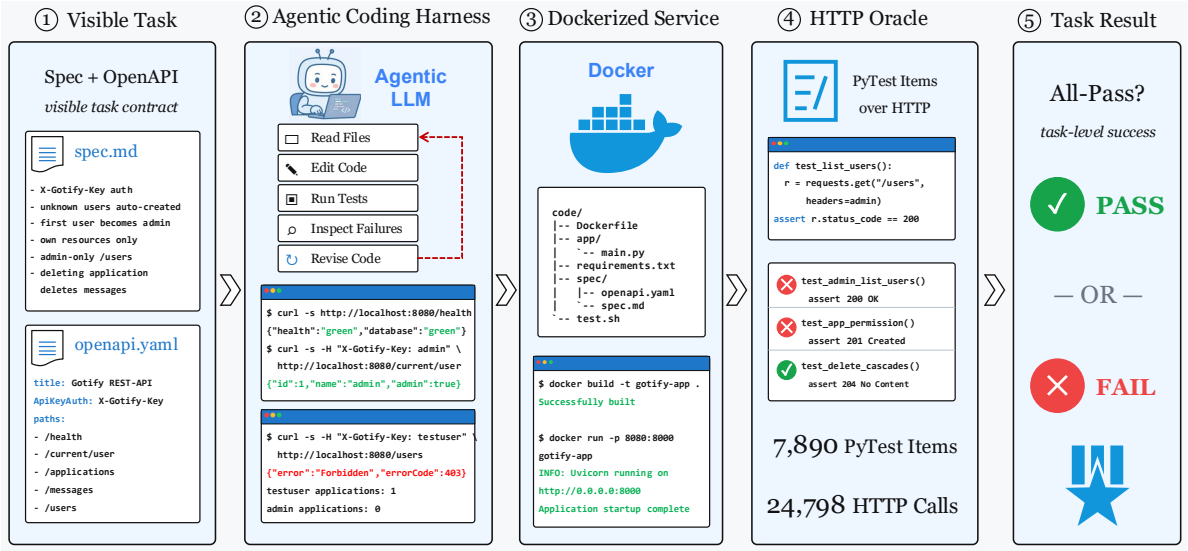

Figure 1: BACKENDFORGE evaluation pipeline. The agent receives only the visible task contract, develops and self-tests a Dockerized backend service, and submits the service for black-box scoring. The evaluator deploys the submitted service and runs the hidden HTTP oracle. Task-level success requires all oracle items to pass.

SWE-bench evaluate whether an LLM can modify an existing codebase to resolve a reported issue, requiring repository understanding and patch generation (Jimenez et al., 2024). Related agentic backend tasks adopt a similar formulation in backend settings, where models implement requested functionality inside existing repositories rather than generating deployable services from scratch (Yang et al., 2026). Other work studies from-scratch application or backend generation (Ran et al., 2025; Vero et al., 2025), but typically evaluates single-turn standalone generation rather than deployable service construction under an agentic coding workflow. Meanwhile, full-stack application benchmarks move closer to end-to-end software generation, but their evaluation often entangles service behavior with frontend rendering, browser automation, GUI interaction, and subjective or unstable interface-level signals (Liu et al., 2026; Tran et al., 2026).

Backend services provide a controlled but realistic substrate for evaluating whether generated software behaves correctly after deployment (Atlidakis et al., 2019; Martin-Lopez et al., 2021). This choice is not meant to reduce end-to-end software generation to backend coding alone. Rather, backend APIs expose much of an application's executable semantics, including interface contracts, persistent state, authentication, authorization, validation, cross-entity side effects, and workflows. They also provide a machine-readable interface that LLMs in agentic workflows can directly invoke through structured requests and responses, reflecting a broader shift toward agent-native software interaction through APIs rather than graphical interfaces (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024). Unlike GUI-based evaluation (Deng et al., 2023), backend behavior can be grounded in explicit specifications, OpenAPI contracts (OpenAPI Initiative, 2024), and deterministic HTTP interactions (Viglianisi et al., 2020). Backend services therefore provide a useful testbed for isolating application-level behavior while still requiring generated code to run as a deployed system.

We introduce BACKENDFORGE, a benchmark for evaluating deployable backend service generation under agentic coding. Each task provides a visible natural-language specification together with an OpenAPI contract, and an LLM must generate a backend service that can be built, deployed, and evaluated through black-box HTTP interactions. BACKENDFORGE contains 56 tasks rewritten from real open-source applications across domains such as e-commerce, content management, project collaboration, notification systems, finance tools, identity management, feature flags, and developer tooling. To reduce direct recoverability from upstream repositories, we rewrite the API contracts, authentication schemes, implementation stacks, and test suites while preserving realistic service semantics.

A central challenge in this setting is constructing an oracle that is strong enough to expose service-

level defects without introducing hidden requirements (Barr et al., 2015; Alonso et al., 2023). Backend services contain many corner cases involving state consistency, authorization isolation, validation, filtering, side effects, and multi-step workflows. Seed test suites rarely cover these behaviors exhaustively, but automatically generated tests can over-specify the task by adding requirements not entailed by the visible specification (Fraser et al., 2015). To address this tension, BACKENDFORGE uses multi-agent oracle co-evolution during benchmark construction. Test generation, specification validation, and reference repair co-evolve under the visible specification and OpenAPI contract. A candidate test is admitted only if it exposes a specification-grounded defect in the reference implementation and the repaired reference passes full regression. This process strengthens oracle coverage of backend corner cases while preserving the task contract given to the model.

Our evaluation shows that current LLMs can implement many local API behaviors, but still struggle to produce complete deployable backend services. We report the success rate, defined as the fraction of tasks for which the generated candidate service passes the final oracle. Across 56 tasks, the strongest model succeeds on only 16 tasks (28.6%). The failures are not dominated by syntax errors or missing endpoints. Instead, models most often fail on service-level execution semantics, including state consistency, authorization isolation, validation, side-effect propagation, and multi-step workflows. These results suggest that deployable backend service generation remains a challenging benchmark for LLMs in agentic coding workflows.

Our contributions are as follows:

- We develop a scalable methodology for constructing specification-grounded backend service oracles, using multi-agent test generation, specification validation, and reference repair to harden HTTP test suites without introducing hidden requirements.
- We introduce BACKENDFORGE, a 56-task benchmark for end-to-end backend service generation from explicit natural-language specifications and OpenAPI contracts, paired with a deterministic black-box HTTP verifier for deployed service behavior.
- We present a systematic empirical evaluation of current LLMs on BACKENDFORGE, characterizing task-level success and recurring semantic failure modes in deployable backend generation.

Benchmark	Rigorous Eval.	Agentic	End-to-End
HumanEval (Chen et al., 2021)	✓	×	×
SWE-bench (Jimenez et al., 2024)	✓	✓	×
WebCoderBench (Liu et al., 2026)	×	✓	✓
Vibe Code Bench (Tran et al., 2026)	×	✓	✓
AppForge (Ran et al., 2025)	×	×	✓
BaxBench (Vero et al., 2025)	✓	×	✓
ABC-Bench (Yang et al., 2026)	✓	✓	×
BACKENDFORGE (ours)	✓	✓	✓

Table 1: Comparison with representative code, application, and backend benchmarks. Rigorous Eval. denotes deterministic scoring. Agentic denotes interaction with a development environment. End-to-End denotes generation of a runnable application or service from a specification.

2 Related Work

BACKENDFORGE differs from prior benchmarks by combining three properties that are usually studied separately: reproducible evaluation, agentic development, and end-to-end runnable output. Table 1 compares representative benchmarks along these three axes. Unlike benchmarks that focus on isolated functions, repository-level patches, or application generation without deterministic scoring, BACKENDFORGE evaluates whether an agent-generated backend service satisfies a visible API contract after deployment under a co-evolved HTTP oracle.

Code generation and repository repair. Code generation benchmarks evaluate functions, classes, or programming-problem solutions under unit tests or execution checks (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Du et al., 2023), with later work strengthening the oracle through augmented or generated tests (Liu et al., 2023; Chen et al., 2022). Repository repair benchmarks such as SWE-bench evaluate patch generation and agent-computer interaction within existing codebases (Jimenez et al., 2024; Yang et al., 2024).

Application and service generation. WebCoderBench, Vibe Code Bench, and AppForge evaluate web or application generation from user-facing requirements (Liu et al., 2026; Tran et al., 2026; Ran et al., 2025). Their evaluation can mix backend logic with frontend rendering, browser workflows, visual correctness, and platform-specific constraints.

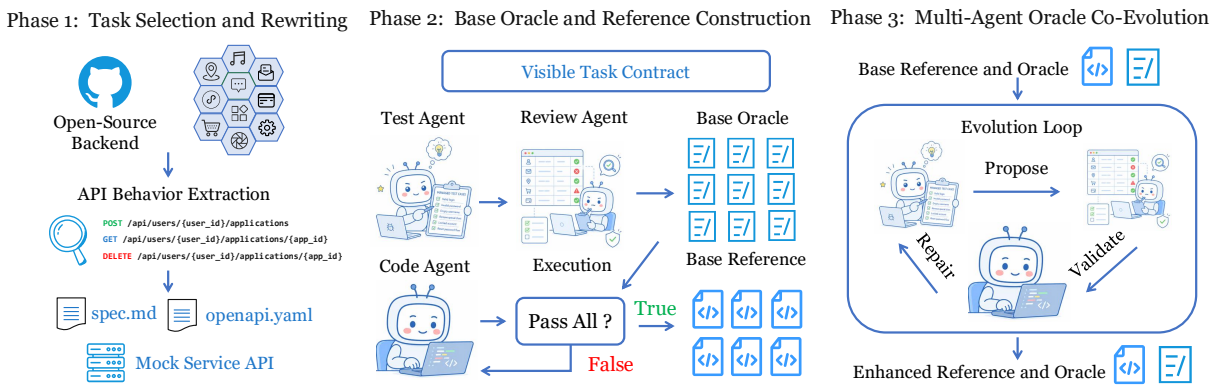

Figure 2: BACKENDFORGE construction pipeline. Source backend applications are rewritten into visible task contracts, which serve as the source of truth for base oracle construction, reference construction, and repair-guided oracle co-evolution. Candidate oracle items are admitted only when they are grounded in the visible contract and expose a defect in the old reference.

Backend generation and agentic backend coding. BaxBench evaluates standalone backend application generation from scenario specifications across frameworks (Vero et al., 2025), and ABC-Bench evaluates agentic backend coding inside existing repositories with environment setup and external API tests (Yang et al., 2026).

3 BACKENDFORGE

BACKENDFORGE is built around three invariants that make end-to-end backend generation measurable. First, every task has a visible contract consisting of a natural-language specification and an OpenAPI file. Second, every scored artifact is a deployed service observed only through HTTP requests. Third, oracle strengthening is allowed only when a candidate test is grounded in the visible contract. Figure 2 summarizes this construction pipeline. Rather than trusting a single LLM-generated artifact, BACKENDFORGE uses LLM agents only in roles where their outputs can be reviewed, executed, and repaired under deterministic feedback.

BACKENDFORGE contains 56 tasks with substantial API surface area and test coverage. Across these tasks, the OpenAPI contracts define 2,345 API operations, where an operation is an HTTP method and path pair such as `GET /users/{id}` or `POST /projects`. The final oracle contains 7,890 pytest items, consisting of 7,250 base oracle items and 640 admitted co-evolved pytest items. A pytest item is one collected pytest test unit after parameterization, and each item issues one or more HTTP requests against the deployed service. In total, the oracle executes 24,798 HTTP request invocations, so BACKENDFORGE measures stateful service behavior rather than isolated function outputs.

3.1 Task Selection and Controlled Rewriting

BACKENDFORGE starts from real open-source backend applications, but each source is rewritten into a controlled benchmark task. We select applications that exercise service-level backend behavior, especially persistence, access control, validation, entity relationships, and multi-step workflows. The resulting task set spans commerce, collaboration, productivity, identity, analytics, and infrastructure-style domains. For more details, see Appendix B. This selection keeps the benchmark close to realistic backend engineering while avoiding tasks that reduce to a single algorithmic routine or a static response template.

Controlled rewriting serves two purposes by preserving realistic service semantics while preventing upstream code from being a drop-in solution. For every task, we rewrite the API surface, authentication model, behavior rules, and evaluation items, then express the rewritten behavior through a visible natural-language specification and an OpenAPI contract. The original application code therefore cannot simply be copied to pass the benchmark. At the same time, the task remains solvable from the released materials because the required behavior is documented in the visible contract rather than hidden in the source repository.

3.2 Visible Task Contract

Each BACKENDFORGE task exposes a visible contract that is the single source of truth for construction and evaluation. The visible materials consist

of a product specification (`spec.md`) and an OpenAPI contract (`openapi.yaml`). The specification describes behavior that is difficult to capture in schemas alone, such as roles, ownership rules, state transitions, and workflow side effects. The OpenAPI contract defines the endpoint surface, request and response schemas, status codes, and authentication mechanism. During construction, every generated test, review decision, and reference repair is checked against these two artifacts rather than against private implementation assumptions.

The visible contract also defines the execution boundary of a task. A submitted solution must provide a Python backend service and a Dockerfile that builds and runs the service in isolation. We restrict the implementation language to Python to control the experimental setting, but leave framework, storage, and internal design choices open. The benchmark does not inspect source code during scoring. It evaluates only HTTP behavior at the service boundary, allowing different internal implementations to be compared under the same behavioral contract.

Tasks that require external integrations use deterministic mock services rather than real third-party APIs. For example, calendar booking, notification delivery, and payment workflows may require the generated service to call a harness-provided HTTP service under `mock_services/`. These mocks are part of the benchmark environment, not part of the candidate output. The visible `spec.md` describes the mock service URL and request/response contract, so the service must integrate with it as it would with an external API. In the current task set, five tasks include such mock services for calendar, delivery, or payment behavior. This design keeps integration-heavy backend semantics in scope while avoiding nondeterministic online dependencies.

3.3 Base Oracle and Reference Construction

The base oracle is constructed before the reference implementation so that the first regression target is not fitted to a particular service. During task rewriting, a test agent receives `spec.md` and `openapi.yaml`, then drafts black-box pytest items for documented endpoint behavior and multi-step workflows. A review agent checks every candidate item against the same two visible materials and rejects items that overinterpret the task, guess undocumented behavior, or rely on speculative business rules. The retained items become the fixed base oracle. Across the 56 tasks, this process produces 7,250 base oracle items that establish broad coverage over endpoint behavior and common workflows.

Only after the base oracle is fixed do we build the reference implementation. A code agent receives `spec.md` and `openapi.yaml` as the implementation target, while the fixed base oracle acts as a regression suite for the current reference. When the reference fails, the construction harness reports the failing pytest item and a concise error summary, allowing the code agent to repair local implementation defects.

3.4 Multi-Agent Oracle Co-Evolution

The base oracle is broad, but it is not treated as complete. Backend correctness often depends on authorization boundaries, state isolation, validation, reference integrity, and workflow side effects that are easy to miss in the initial base oracle construction. At the same time, automatically adding harder tests is unsafe because a plausible backend behavior is not necessarily a specified behavior. For example, one API may return `404` when deleting a missing resource, while another treats deletion as idempotent and returns `204`. If the visible contract does not choose one behavior, a pytest item that enforces either choice introduces a hidden requirement. The co-evolution loop addresses this tension by searching for useful missing tests while using review and repair to prevent unsupported requirements from entering the oracle.

The co-evolution loop uses the current reference as a probe for missing oracle coverage. Given the visible contract, the current reference implementation, and the already accepted items, the test agent proposes a candidate black-box pytest item and executes it against the old reference. If the candidate passes, it is discarded in this loop because it does not reveal a reference defect. If the candidate fails, the candidate and its reference-execution trace are passed to the review agent. This failure-first filter makes the loop scalable. Many candidates can be generated, but only candidates that expose concrete reference behavior receive further attention.

The review agent turns a failing candidate into either a rejected assumption or an eligible reference defect. It checks whether the candidate has traceable support in `spec.md` or `openapi.yaml`, and rejects items that depend on undocumented behavior, ambiguous status-code choices, or harness mistakes. The remaining failures are treated as

LLM (Thinking Effort)	Size	Base SR	+R1 SR	+R2 SR	Final SR	Δ SR	Cost/Task	Output Tokens/Task
Proprietary LLMs								
GPT-5.5 (xhigh)	N/A	55.36%	**42.86%**	**37.50%**	**28.57%**	-48.39%	$3.39	51.06K
Claude Opus 4.7 (max)	N/A	**58.93%**	37.50%	28.57%	17.86%	-69.70%	$5.55	75.38K
Gemini 3.5 Flash	N/A	21.43%	10.71%	8.93%	5.36%	-75.00%	$2.15	61.30K
Claude Sonnet 4.6 (max)	N/A	39.29%	8.93%	5.36%	3.57%	-90.91%	$2.76	65.17K
Qwen 3.6 Max (high)	N/A	23.21%	5.36%	1.79%	1.79%	-92.31%	$0.73	42.04K
Qwen 3.6 Plus (high)	N/A	23.21%	1.79%	0.00%	0.00%	-100.00%	$0.33	43.91K
Open-Source LLMs								
DeepSeek V4 Pro (max)	1.6T-A49B	32.14%	12.50%	7.14%	3.57%	-88.89%	$0.09	50.85K
GLM 5.1	754B-A40B	26.79%	8.93%	3.57%	3.57%	-86.67%	$1.31	54.82K
Kimi K2.6	1T-A32B	39.29%	12.50%	3.57%	1.79%	-95.45%	$0.77	52.54K
DeepSeek V4 Pro (high)	1.6T-A49B	25.00%	8.93%	3.57%	1.79%	-92.86%	$0.07	42.34K
DeepSeek V4 Flash (high)	284B-A13B	21.43%	3.57%	0.00%	0.00%	-100.00%	$0.03	40.24K
DeepSeek V4 Flash (max)	284B-A13B	19.64%	8.93%	1.79%	0.00%	-100.00%	$0.04	53.25K
Hunyuan Hy3 (high)	295B-A21B	12.50%	3.57%	0.00%	0.00%	-100.00%	$0.27	48.79K
MiniMax M2.7	229B-A10B	5.36%	1.79%	0.00%	0.00%	-100.00%	$0.27	48.50K

Table 2: Main BACKENDFORGE results. SR denotes task-level all-pass success rate over the same 56-task denominator. +R1 and +R2 cumulatively add accepted co-evolved items to the base oracle; the final oracle adds all accepted co-evolved items. Δ is the signed relative change from Base SR to Final SR, computed as $(\text{Final SR} - \text{Base SR})/\text{Base SR}$; red negative values indicate relative decreases. Closed LLMs and LLMs with undisclosed sizes use N/A in the Size column. A denotes active parameters for MoE LLMs.

genuine reference defects. This review step is necessary because failure on the old reference alone is not evidence that a test belongs in the benchmark.

Confirmed reference defects are then passed to the code agent for constrained repair. The code agent may modify only the task reference implementation, not the specification, OpenAPI contract, or reviewed candidate tests. Once a candidate item has passed review, it is treated as a valid specification-grounded requirement, and the reference is repaired until it satisfies this item together with the base oracle and all previously accepted evolved items. After each repair attempt, the harness runs the full regression suite against the patched reference. On each failed attempt, the harness reports the failing pytest item and a concise error summary to guide the next repair. The item is admitted only after the repaired reference passes this full regression suite, ensuring that oracle growth preserves all existing accepted behavior.

Accepted items are accumulated to form the co-evolved part of the final oracle. The final BACKENDFORGE artifact contains the rewritten visible contracts, deterministic mock services when needed, internal reference implementations used during construction, and the final oracle. The final oracle is the union of the 7,250 base oracle items and 640 admitted co-evolved items. This construction uses LLM agents where their outputs can be checked locally and repaired under regression, while the benchmark boundary remains deterministic. Only reviewed, contract-grounded, regression-preserving items become part of the scored oracle.

4 Evaluation

4.1 Experimental Setup

We select frontier agentic LLMs from both proprietary and open-source families, and evaluate all of them using the same mini-SWE-style agentic coding harness based on the agent-computer-interface design principles studied by SWE-agent (Yang et al., 2024). The proprietary models are GPT-5.5 (OpenAI, 2026), Claude Opus 4.7 (Anthropic, 2026b), Gemini 3.5 Flash (Google DeepMind, 2026), Claude Sonnet 4.6 (Anthropic, 2026c), Qwen 3.6 Max (Qwen Team, 2026a), and Qwen 3.6 Plus (Qwen Team, 2026b). The open-source models are DeepSeek V4 Pro (DeepSeek-AI, 2026), GLM 5.1 (GLM-5 Team, 2026), Kimi K2.6 (Moonshot AI, 2026), DeepSeek V4 Flash (DeepSeek-AI, 2026), Hunyuan Hy3 (Tencent Hy Team, 2026), and MiniMax M2.7 (MiniMax AI, 2026).

We report task-level success rate (SR) as the primary metric. For each oracle, a task is counted as successful only if the generated service passes all required pytest items for that task.

4.2 Main Results

Current LLMs generate many locally correct backend behaviors, but they rarely construct fully correct services. Table 2 reports task-level success rates over the same 56 tasks. Under the base oracle, the strongest LLMs solve a substantial fraction of tasks: Claude Opus 4.7 reaches

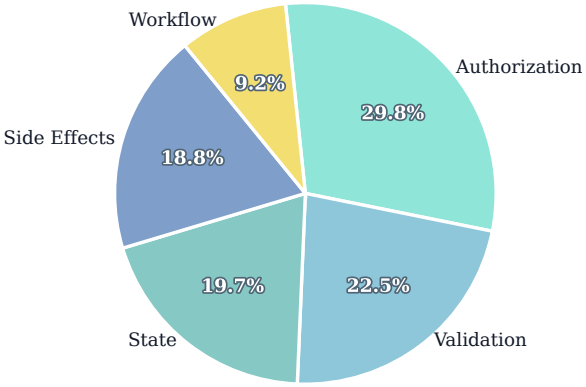

Figure 3: Semantic distribution of accepted co-evolved pytest items in the final oracle. Each accepted co-evolved pytest item is assigned to one primary category using its recorded bug-pattern metadata.

Category	Base	Base %	Evol.	Evol. %	Δ pp
API surface	2,636	36.4	0	0.0	−36.4
Authorization	1,041	14.4	191	29.8	+15.5
Validation	2,248	31.0	144	22.5	−8.5
State	177	2.4	126	19.7	+17.2
Side effects	709	9.8	120	18.8	+9.0
Workflow	439	6.1	59	9.2	+3.2
Total	7,250	100.0	640	100.0	–

Table 3: Distributional contrast between base oracle and accepted co-evolved pytest items. Δ is Evolved % minus Base %.

58.9%, and GPT-5.5 reaches 55.4%. Under the final oracle, however, success drops sharply. GPT-5.5 solves 16 of 56 tasks (28.6%), Claude Opus 4.7 solves 10 (17.9%), and every other configuration solves at most 3. This gap shows that endpoint-level progress is real but incomplete. Generated services often handle many routes and workflows while still failing one required invariant, validation rule, or cross-entity side effect.

Top proprietary models provide the strongest final-oracle performance at substantially higher generation cost. GPT-5.5 and Claude Opus 4.7 solve 16/56 and 10/56 tasks under the final oracle, while every other configuration solves at most 3/56. This performance advantage comes with higher average cost per task. GPT-5.5 costs $3.39 and Claude Opus 4.7 costs $5.55 in our runs, compared with sub-dollar costs for several open-source configurations.

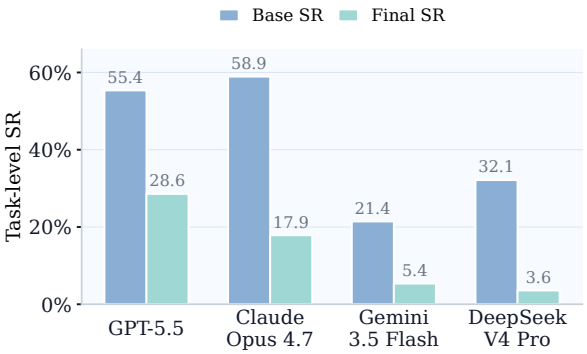

Figure 4: Success-rate drop from the base oracle to the final oracle for representative LLMs.

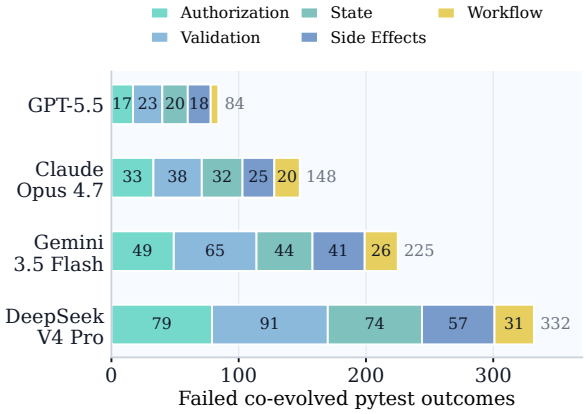

Figure 5: Distribution of error types across models.

5 Analysis

5.1 A small hardened oracle has a large effect

The hardened oracle adds relatively few pytest items, but these items have a large effect on task-level success. Table 3 shows that the final oracle contains 640 accepted co-evolved pytest items in addition to the 7,250 base oracle items. This is a modest 8.8% increase in item count, yet Table 2 shows a large decrease in all-pass success once those items are included. The added items are not random extra coverage. Figure 3 shows that they concentrate on authorization, validation, and stateful workflow behavior.

The retained co-evolved items reveal failures that the base oracle misses. Figure 4 shows monotonic attrition from the base oracle to the final oracle. Claude Opus 4.7 drops from 58.9% to 17.9%, while GPT-5.5 drops from 55.4% to 28.6%. These drops indicate that the accepted co-evolved items capture specification-grounded behaviors beyond broad endpoint coverage. Complete backend success therefore requires satisfying many small contract semantics simultaneously.

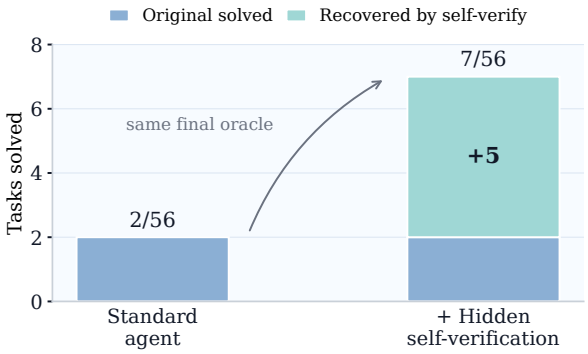

Figure 6: Hidden self-verification improves DeepSeek V4 Pro on the final oracle.

The remaining failures cover multiple backend contract categories. Figure 5 shows that validation is the largest category across models, ranging from 23 failed outcomes for GPT-5.5 to 91 for DeepSeek V4 Pro. Authorization and state failures are also substantial, with DeepSeek V4 Pro failing 79 authorization outcomes and 74 state outcomes. Thus, final-oracle attrition reflects broad difficulty with ordinary service contracts rather than one narrow failure mode.

5.2 Review Agent Prevents Hidden Requirements

The review agent is essential to making the hardened oracle a stable benchmark signal rather than merely a harder test suite. It reviews candidate pytest items only after the test agent has shown that they fail on the current reference implementation. The review agent then checks the failing candidates against the visible specification and OpenAPI contract, rejecting items that require behavior not uniquely supported by those materials. In final pool construction, it filtered 113 candidate pytest items. The most common reasons were task overinterpretation, unsupported assertions, and written-contract gaps.

5.3 Hidden Self-Verification Helps but Does Not Solve the Benchmark

Sparse hidden feedback improves final-oracle success. We evaluate a stronger protocol in which the agent can call `backendforge_self_verify` during development. The tool builds the current candidate service and runs the final oracle, but returns only `SELF_VERIFY_PASS` or `SELF_VERIFY_FAIL` with a coarse failure phase such as build, startup, or tests. It does not expose pytest node IDs, assertions, hidden requests, test source code, the reference implementation, or construction-time traces. Under this protocol, DeepSeek V4 Pro solves 7 of 56 tasks under the final oracle, compared with 2 of 56 in the main protocol. As shown in Figure 6, the original two successes are retained, and self-verification recovers five additional tasks.

```
[verifier call 9]
SELF_VERIFY_FAIL
Phase: tests
[LLM thinking]
I've tried many fixes but I'm still failing...
OK, I've done everything I can think of.
I'm going to stop here and submit.
[tool call]
echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT
```

Figure 7: Representative DeepSeek V4 Pro self-verification trace. In the `maybefinance` task, the agent receives a ninth test-phase `FAIL` and submits without a `PASS` signal, despite remaining verifier budget.

Agents may abandon exploration even when hidden feedback remains available. Figure 7 shows a representative failure trajectory. After nine failed verifier calls, DeepSeek V4 Pro recognizes that the service is still failing, exhausts its own debugging hypotheses, and submits without receiving a `SELF_VERIFY_PASS` signal. Thus, sparse hidden feedback changes the development protocol and can recover some failures, but it does not collapse the benchmark. 49 of 56 tasks still fail for this model even with self-verification.

6 Conclusion

As LLM coding workflows become increasingly agentic, evaluation must move beyond isolated code generation toward deployable end-to-end software artifacts. BACKENDFORGE approaches this problem through backend contract realization, where a visible specification and OpenAPI contract define a deployable service evaluated through deterministic black-box HTTP interactions. Using a multi-agent oracle co-evolution framework, we strengthen backend evaluation without introducing hidden requirements. Our results show that current LLMs can implement many local API behaviors, but still struggle with core backend semantics such as validation, authorization, state consistency, side effects, and workflows. Additional experiments indicate that self-verification can partially mitigate these failures, but most tasks remain unsolved, suggesting that deployable backend generation remains a challenging problem for agentic LLMs.

Limitations and Potential Risks

BACKENDFORGE is scoped to Python backend services evaluated through deterministic black-box HTTP tests. This scope makes the benchmark controllable and reproducible, but it does not cover full-stack user interfaces, other implementation stacks, performance engineering, deployment security, or production hardening.

The final oracle is strengthened through repair-guided co-evolution, but it is still bounded by the visible task contracts and by the admitted test items. Although accepted co-evolved items are reviewed for contract grounding and regression preservation, the oracle cannot exhaustively cover all possible backend behaviors. A model that passes BACKENDFORGE should therefore not be assumed to produce production-ready services without additional review, security analysis, and deployment-specific testing.

The benchmark also carries potential dual-use risks. Improving agentic backend generation may help automate useful software development, but the same capability could be misused to generate backend services for spam, abuse infrastructure, deceptive applications, or poorly secured data-handling systems. For this reason, BACKENDFORGE evaluates functional correctness under visible contracts rather than endorsing unrestricted autonomous deployment. Systems built with such agents should include human oversight, security checks, access-control review, and safeguards before being connected to real users, sensitive data, or external services.

References

Juan Carlos Alonso, Sergio Segura, and Antonio Ruiz-Cortés. 2023. AGORA: Automated generation of test oracles for REST APIs. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1018–1029.

Anthropic. 2026a. Claude Code Docs: Overview. https://code.claude.com/docs/en/overview.

Anthropic. 2026b. Claude opus 4.7 system card. https://www.anthropic.com/claude-opus-4-7-system-card.

Anthropic. 2026c. Claude sonnet 4.6 system card. https://www.anthropic.com/claude-sonnet-4-6-system-card.

Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2019. RESTler: Stateful REST API fuzzing. In *Proceedings of the 41st International Conference on Software Engineering*, pages 748–758.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. CodeT: Code generation with generated tests. *Preprint*, arXiv:2207.10397.

Jinyang Chen, Binyuan Chen, Haoran Zhang, Xiaoyu Yang, Yilun Deng, Hongwei Jiang, Seung-won Lee, Xuan Zhang, Minjoon Kim, Hwanjun Yu, Jian Pei, Fei Wang, Chi Wang, and Ahmed Hassan Awadallah. 2024. TaskWeaver: A code-first agent framework. *Preprint*, arXiv:2311.17541.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

DeepSeek-AI. 2026. DeepSeek-V4: Towards highly efficient million-token context intelligence. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2023. ClassEval: A manually-crafted benchmark for evaluating LLMs on class-level code generation. *Preprint*, arXiv:2308.01861.

Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C. Briand. 2021. Using domain-specific corpora for improved handling of ambiguity in requirements. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*, pages 1485–1497.

Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. 2015. Does automated unit test generation really help software testers? a controlled empirical study. *ACM Transactions on Software Engineering and Methodology*, 24(4):1–49.

GLM-5 Team. 2026. GLM-5: from vibe coding to agentic engineering. https://arxiv.org/abs/2602.15763. *Preprint*, arXiv:2602.15763.

Google DeepMind. 2026. Gemini 3.5 flash model card. https://deepmind.google/models/model-cards/gemini-3-5-flash.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Advances in Neural Information Processing Systems*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2024. AI agents that matter. *Preprint*, arXiv:2407.01502.

Chenxu Liu, Yingjie Fu, Wei Yang, Ying Zhang, and Tao Xie. 2026. WebCoderBench: Benchmarking web application generation with comprehensive and interpretable evaluation metrics. *Preprint*, arXiv:2601.02430.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. *Preprint*, arXiv:2305.01210.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, and 3 others. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. *Preprint*, arXiv:2102.04664.

Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2021. RESTest: Automated black-box testing of RESTful web APIs. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 682–685.

MiniMax AI. 2026. MiniMax M2.7 model card. https://huggingface.co/MiniMaxAI/MiniMax-M2.7.

Moonshot AI. 2026. Kimi-K2.6 model card. https://huggingface.co/moonshotai/Kimi-K2.6.

OpenAI. 2026. GPT-5.5 system card. https://openai.com/index/gpt-5-5-system-card/.

OpenAPI Initiative. 2024. OpenAPI specification version 3.1.1. https://spec.openapis.org/oas/v3.1.1.html.

OpenCode. 2026. OpenCode: The Open Source AI Coding Agent. https://opencode.ai/.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 15174–15186.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.

Qwen Team. 2026a. Qwen3.6-max-preview: Smarter, sharper, still evolving. https://qwen.ai/blog?id=qwen3.6-max-preview.

Qwen Team. 2026b. Qwen3.6-plus: Towards real world agents. https://qwen.ai/blog?id=qwen3.6.

Dezhi Ran, Yuan Cao, Mengzhou Wu, Simin Chen, Yuzhe Guo, Jun Ren, Zihe Song, Hao Yu, Jialei Wei, Linyi Li, Wei Yang, Baishakhi Ray, and Tao Xie. 2025. AppForge: From assistant to independent developer – are GPTs ready for software development? *Preprint*, arXiv:2510.07740.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Tencent Hy Team. 2026. Hy3-preview model card. https://huggingface.co/tencent/Hy3-preview.

Hung Tran, Langston Nashold, Rayan Krishnan, Antoine Bigeard, and Alex Gu. 2026. Vibe code bench: Evaluating AI models on end-to-end web application development. *Preprint*, arXiv:2603.04601.

Mark Vero, Niels Mündler, Victor Chibotaru, Veselin Raychev, Maximilian Baader, Nikola Jovanović, Jingxuan He, and Martin Vechev. 2025. BaxBench: Can LLMs generate correct and secure backends? *Preprint*, arXiv:2502.11844.

Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. 2020. RESTTESTGEN: Automated black-box testing of RESTful APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification*, pages 142–152.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Conference on Language Modeling*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*.

Jie Yang, Honglin Guo, Li Ji, Jiazheng Zhou, Rui Zheng, Zhikai Lei, Shuo Zhang, Zhiheng Xi, Shichun Liu, Yuxin Wang, Bo Wang, Yining Zheng, Tao Gui, and Xipeng Qiu. 2026. ABC-bench: Benchmarking agentic backend coding in real-world development. *Preprint*, arXiv:2601.11077.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. *Preprint*, arXiv:2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*.

A Agent Prompt Templates

This appendix reports the prompt templates used by the construction agents in BACKENDFORGE. The templates are lightly normalized for presentation: task-specific content, previous-round summaries, and failure reports are shown as placeholders, while operational details unrelated to the paper's construction logic are omitted. The three prompts reflect the roles in Section 3: the test agent proposes black-box pytest items, the code agent implements or repairs the reference service, and the review agent decides whether a candidate test is grounded in the visible contract.

A.1 Test Agent

The test agent receives the visible task contract and construction history, then proposes candidate pytest items. The prompt emphasizes black-box HTTP behavior and requires every assertion to be traceable to `spec.md` or `openapi.yaml`.

Listing 1: Test agent prompt template.

```
You are the BackendForge test agent.

Goal:
Generate adversarial black-box HTTP pytest
    tests for a backend service
described by spec.md and openapi.yaml.

Authority:
- Use only the visible specification and
    OpenAPI contract.
- Do not invent hidden requirements.
- Every assertion must be traceable to spec.md
     or openapi.yaml.

Focus on backend contract failures:
1. field naming or response-shape mismatch
2. missing nested relation data
3. declared filters, sorting, or pagination
    not implemented
4. incorrect HTTP status codes
5. empty or fake business logic
6. authorization or data-isolation failure
7. cascade, side-effect, or preservation
    failure
8. uniqueness, ordering, idempotency, or state-
    machine failure

Inputs:
- Task name: {{ task_name }}
- spec.md: {{ spec_md }}
- openapi.yaml: {{ openapi_yaml }}
- Existing base tests: {{
    existing_tests_summary }}
- Previously accepted co-evolved tests: {{
    accepted_tests_summary }}
- Previously rejected candidate tests: {{
    rejected_tests_summary }}
- Reference/testcase repair decisions: {{
    reference_decision_summary }}
- Round guidance: {{ round_guidance }}

Rules:
- Use HTTP requests against BASE_URL.
- Do not inspect source code, database files,
    logs, or internal state.
- Avoid arbitrary timing assumptions and
    nondeterministic ordering.
- Prefer tests likely to catch realistic LLM-
    generated backend bugs.
- Do not repeat a previously rejected
    unsupported assertion.
- If prior accepted tests passed, search for
    deeper workflows, negative cases,
  ownership matrices, state transitions, and
    side-effect invariants.

Return only JSON:
{
  "candidates": [
    {
      "test_name": "...",
      "level": "L2 or L3",
      "bug_patterns": ["
    authorization_isolation"],
      "spec_traces": ["exact spec quote or
    section"],
      "openapi_traces": ["GET /api/path
    response schema"],
      "rationale": "...",
      "expected_reference_behavior": "...",
      "expected_generated_failures": ["
    run_name: reason"],
      "test_code": "standalone pytest code
    using requests and BASE_URL"
    }
  ]
}
```

A.2 Code Agent

The code agent is used during benchmark construction to build and repair the reference service. Its prompt separates implementation from oracle definition: the agent may repair only the reference implementation, while the visible contract and accepted tests remain fixed.

Listing 2: Code agent prompt template for reference implementation and repair.

```
You are the BackendForge code agent.

Goal:
Implement or repair the task reference backend
     so that it satisfies the visible
contract in spec.md and openapi.yaml.

Authority:
- The visible specification and OpenAPI
    contract define the required behavior.
- Test failures are debugging evidence, not
    permission to hardcode a testcase.
- If a requested behavior is not supported by
    the visible contract, mark it for
  review instead of patching around it.

Allowed write scope:
```

```
- tasks/{{ task_name }}/reference/**

Forbidden changes:
- Do not modify spec.md.
- Do not modify openapi.yaml.
- Do not modify existing tests, candidate
    tests, or co-evolution artifacts.
- Do not depend on generated model outputs.

Implementation requirements:
- Build a complete backend service that
    listens on port 8000.
- Initialize required storage on startup, but
    do not insert seed data.
- Match documented endpoints, parameters,
    status codes, and response schemas.
- Implement general service behavior, not
    special cases for candidate values,
  test function names, UUIDs, emails, or
    resource names.
- Preserve authorization, ownership,
    validation, uniqueness, workflow, and
  side-effect semantics required by the
    visible contract.

Repair inputs:
- Task name: {{ task_name }}
- spec.md: {{ spec_md }}
- openapi.yaml: {{ openapi_yaml }}
- Repair batch: {{ repair_batch_json }}

After repair, all of the following must pass:
1. original base oracle tests
2. previously accepted co-evolved tests
3. current candidate tests in the repair batch
4. fresh-container regression checks

Output:
- files changed
- claims or failures addressed
- behavior implemented
- regression commands run
- any candidate that should be reclassified as
    SPEC_GAP or TESTCASE_BUG
```

A.3 Review Agent

The review agent protects the benchmark from hidden requirements. It reviews the behavioral claim of each candidate test rather than incidental setup code, and keeps only tests whose central assertion is directly supported by the visible contract or is an unavoidable consequence of it.

Listing 3: Review agent prompt template.

```
You are the BackendForge review agent.

Goal:
Decide whether each candidate pytest item is
    grounded in the visible backend
contract and can be admitted to the benchmark
    oracle.

Authority:
- Use only spec.md and openapi.yaml.
- Do not use the reference implementation as a
    source of requirements.
- Do not use generated implementation behavior,
    product knowledge, framework
  conventions, or subjective engineering
    preferences as requirements.

Review standard:
- Review the behavioral purpose of the
    candidate, not incidental setup code.
- KEEP only if the asserted requirement is
    directly stated by spec.md,
  directly stated by openapi.yaml, or an
    unavoidable consequence of an explicit
  contract.
- FILTER if the candidate relies on plausible
    but undocumented behavior, an
  ambiguous status code, unspecified ordering,
    default pagination behavior,
  unstated cascade effects, implementation
    details, or a stronger rule than the
  contract supports.
- Do not filter merely because a test is
    negative or strict. If the rejection
  rule is explicit, checking that a rejected
    request creates no successful
  mutation is allowed.

Inputs:
- Task name: {{ task_name }}
- spec.md: {{ spec_md }}
- openapi.yaml: {{ openapi_yaml }}
- Candidate tests: {{ candidates_json }}

Return only JSON:
{
  "task": "{{ task_name }}",
  "decisions": [
    {
      "candidate_id": "string",
      "verdict": "KEEP or FILTER",
      "category": "SUPPORTED |
    OVERINTERPRETATION | SPEC_GAP |
    UNSUPPORTED_ASSERTION | TEST_BUG",
      "confidence": 0.0,
      "reason": "short concrete reason",
      "unsupported_assertions": ["only for
    FILTER"],
      "spec_support": ["visible spec/openapi
    support or relevant missing support"]
    }
  ]
}
```

B Benchmark Details

Tables 4–7 list the 56 tasks used in the final BACKENDFORGE evaluation set. Each row gives the rewritten task name, coarse domain, and intended backend service boundary exposed through `spec.md` and `openapi.yaml`. Source applications were selected from publicly available open-source repositories and rewritten into benchmark tasks. We intend to release the benchmark artifacts under an appropriate research-friendly license while respecting upstream licenses.

Task	Domain(s)	Description
Actual Budget	Personal finance, budgeting	Envelope budgeting service for accounts, transactions, monthly budgets, payees, scheduled transactions, and user-scoped financial data.
BookStack	Knowledge base, content management	Hierarchical wiki service with shelves, books, chapters, pages, page revisions, attachments, search, and permission controls.
Cal.com	Scheduling, calendar integration	Scheduling platform where users define availability, event types, teams, and bookings that interact with a mock calendar service.
Discourse	Community forum, moderation	Forum backend with categories, topics, posts, reactions, private messages, badges, and trust-level permission rules.
Django CRM	CRM, sales pipeline	Customer relationship management API for contacts, companies, leads, opportunities, activities, and pipeline stages.
Documenso	Document signing, workflow	Document signing service for documents, recipients, signature fields, sending, signing events, templates, and audit trails.
Dub	Link management, analytics	Short-link platform with workspaces, custom domains, tags, folders, click analytics, and role-scoped team collaboration.
EZBookkeeping	Personal finance, expense tracking	Expense tracking service for accounts, categories, tags, transactions, monthly summaries, budgets, transfers, and recurring entries.
FitTrackee	Fitness tracking, activity logging	Activity tracker for workouts, personal records, equipment, comments, GPX-like route points, and user-scoped statistics.
Flagsmith	Feature flags, remote config	Feature flag service for organizations, projects, environments, segments, identities, traits, and percentage rollout rules.
Focalboard	Project management, kanban	Kanban-style project management API for boards, cards, views, custom properties, comments, and workspace collaboration.
Forem	Developer community, publishing	Developer community platform for articles, comments, follows, tags, reactions, listings, and organization membership.
Formbricks	Surveys, audience management	Survey platform for question definitions, survey publishing, contacts, responses, segments, teams, and role-based access.
Ghostfolio	Portfolio tracking, wealth management	Wealth management backend for accounts, portfolio activities, holdings, benchmarks, dividends, cash flows, and performance summaries.

Table 4: BACKENDFORGE task list. Each row gives the task name, coarse application domain, and a one-sentence description of the required backend service.

Task	Domain(s)	Description
Gotify	Push notifications, admin management	Self-hosted push notification server with applications, clients, messages, user auto-creation, admin-only user management, and priority filtering.
GrowthBook	Feature flags, experiments	Experimentation platform for feature flags, environments, attributes, A/B experiments, metrics, snapshots, and analysis results.
Healthchecks	Monitoring, cron checks	Background job monitoring service where checks receive pings, track status, schedule grace periods, and trigger notification channels.
HedgeDoc	Markdown collaboration, access control	Collaborative markdown editor backend for notes, permissions, revisions, groups, comments, and share links.
Homebox	Home inventory, maintenance	Home inventory service for locations, labels, items, maintenance logs, attachments, groups, and user-scoped household data.
Hoppscotch	API tooling, collaboration	API development workspace for request collections, environments, teams, variables, history, and shared API testing artifacts.
InvenTree	Inventory, manufacturing	Inventory management system for parts, stock, suppliers, purchase orders, bills of materials, build orders, and stock movements.
Invoice Ninja	Invoicing, small-business finance	Invoicing platform for clients, quotes, invoices, payments, expenses, time tracking, products, and recurring billing records.
KitchenOwl	Recipes, grocery planning	Household recipe and grocery service with shared shopping lists, meal plans, recipes, pantry items, and expense tracking.
Lago	Billing, subscriptions	Usage-based billing API for customers, plans, subscriptions, meters, usage events, invoices, payments, and coupons.
Langfuse	LLM observability, evaluation	Observability backend for LLM traces, generations, prompts, datasets, evaluation scores, projects, and API keys.
Leantime	Project management, OKRs	Goal-focused project management service with goals, milestones, tasks, sprints, timesheets, ideas, and project-level permissions.
Linkwarden	Bookmark management, collaboration	Collaborative bookmark manager for collections, links, tags, archive metadata, sharing, and user or team access controls.
Logto	Identity management, authorization	Identity platform management API for users, applications, API resources, permissions, roles, organizations, and sign-in settings.

Table 5: BACKENDFORGE task list, tasks 15–28.

Task	Domain(s)	Description
Maybe Finance	Personal finance, analytics	Personal finance dashboard for accounts, transactions, categories, budgets, goals, net-worth views, and spending insights.
RecipeVault	Recipes, meal planning	Self-hosted recipe manager inspired by Mealie with recipes, categories, tags, meal plans, shopping lists, ratings, and households.
Medusa	E-commerce, order management	Headless commerce backend for products, customers, carts, orders, payments, discounts, shipping, and regional configuration.
Memos	Notes, social publishing	Lightweight note-taking service for markdown memos, tags, visibility, resources, reactions, comments, and user timelines.
Miniflux	RSS reader, content aggregation	RSS reader backend for feeds, categories, entries, bookmarks, read state, subscriptions, and feed refresh behavior.
NetBox	Infrastructure management, IPAM	Network and datacenter management API for sites, racks, devices, interfaces, cables, prefixes, IP addresses, and tenants.
NocoDB	No-code database, spreadsheet	No-code database API for bases, tables, typed columns, records, views, filters, comments, webhooks, and audit logs.
Novu	Notifications, workflow orchestration	Notification infrastructure backend for workflows, subscribers, topics, templates, and mock multi-channel delivery behavior.
Ntfy	Push notifications, pub/sub	Topic-based notification service for publishing messages, subscriptions, priorities, attachments, tokens, and access policies.
Outline	Knowledge base, document collaboration	Collaborative wiki backend for collections, nested documents, comments, search, sharing, groups, and access permissions.
Paperless-ngx	Document management, metadata	Document management service for correspondents, document types, tags, custom fields, saved views, tasks, and bulk edits.
Plane	Project management, issue tracking	Project management backend for workspaces, projects, workflow states, issues, cycles, modules, views, pages, and comments.
Plausible	Web analytics, event tracking	Privacy-first analytics API for sites, pageviews, custom events, goals, aggregate metrics, and API-key-scoped reports.
Saleor	E-commerce, inventory and checkout	Multi-channel commerce backend for products, attributes, warehouses, stock, checkouts, payments, orders, and fulfillment flows.

Table 6: BACKENDFORGE task list, tasks 29–42.

Task	Domain(s)	Description
Shlink	URL shortener, analytics	Self-hosted URL shortener with short codes, redirects, tags, visit tracking, domains, QR codes, and rule-based access.
Solidtime	Time tracking, reporting	Time-tracking service for organizations, projects, tasks, time entries, approvals, clients, and aggregate reports.
Spree	E-commerce, multi-store checkout	Multi-store commerce platform with products, taxonomies, carts, orders, promotions, shipping, stock, and checkout state transitions.
Strapi	Headless CMS, content modeling	Headless CMS API for dynamic content types, entries, relations, components, media, localization, and role-based permissions.
Tandoor Recipes	Recipes, meal planning	Recipe management backend for recipes, ingredients, meal plans, shopping lists, nutrition metadata, and cookbook organization.
TaskCafe	Project management, kanban	Kanban project management service for projects, task groups, tasks, labels, checklists, comments, and member assignments.
Traduora	Translation management, localization	Localization platform for projects, locales, translation keys, translated values, collaborators, exports, and progress tracking.
TubeArchivist	Media archiving, video metadata	Media archive API for videos, channels, playlists, download queues, comments, subtitles, sponsor segments, and metadata search.
Twenty	CRM, workspace data model	Workspace CRM backend for people, companies, opportunities, pipelines, custom objects, activities, notes, and team roles.
Umami	Web analytics, tracking	Privacy-focused analytics service for websites, public event collection, sessions, metrics, reports, and team-scoped dashboards.
Vaultwarden	Password manager, encrypted vault	Bitwarden-compatible vault service for ciphers, folders, collections, organizations, sends, attachments, and token-based access.
Vikunja	Task management, collaboration	Self-hosted task management API for projects, nested tasks, labels, due dates, reminders, sharing, and comments.
Zitadel	Identity management, access control	Multi-tenant identity platform for organizations, users, sessions, projects, roles, MFA, password policies, and OAuth-style resources.
Zulip	Team chat, topic-based messaging	Topic-based team chat service with streams, topics, messages, reactions, users, subscriptions, private messages, and moderation rules.

Table 7: BACKENDFORGE task list, tasks 43–56.