2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)

# APIDA-Chat: Structured Synthesis of API Search Dialogues to Bootstrap Conversational Agents

Zachary Eberhart and Collin McMillan
*Department of Computer Science*
*University of Notre Dame*
Notre Dame, IN, USA
{zeberhar, cmc}@nd.edu

***Abstract*—Large-language-model assistants are suitable for explaining popular APIs, yet they falter on niche or proprietary libraries because the multi-turn dialogue data needed for fine-tuning are scarce. We present *APIDA-Chat*, an open-source pipeline that converts symbolic dialogue-act "scripts" into realistic, domain-grounded API Search conversations using a lightweight model for inexpensive training data generation. Phase I pairs a legacy dialogue planner with a high-capability teacher LLM (o4-mini) to synthesize a "gold set" of realized dialogues; then, a smaller Llama 3.2 3B student model is fine-tuned on this corpus. Phase II drops the teacher and reuses the same planner with the fine-tuned model, allowing rapid, low-cost synthesis of new dialogues without exposing source code to external services. The fine-tuned student improves BLEU from 0.38 to 0.50 and BERTScore from 0.88 to 0.91 versus the base model while running entirely on a single consumer GPU. All components are modular and publicly released to serve as a conservative baseline for future work. *APIDA-Chat* is open-sourced at https://github.com/Zeberhart/apida-chat and a video demo is available at https://youtu.be/YqmZBHyGbPs .**

***Index Terms*—API Search, Dialogue Management, Synthetic Data Generation, Large Language Models (LLMs)**

## I. Introduction

Conversational assistants are rapidly becoming a standard interface for programmers who need to search APIs, debug code snippets, or explore unfamiliar libraries. A recent Stack-Overflow survey demonstrated that over 60% of developers already rely on conversational AI tools, and over 70% of those are currently using or interested in using those tools to learn about a codebase [1]; qualitative studies echo these findings, identifying API exploration as a top use-case for AI assistants [2]. These chatbots rely on large language models (LLMs) that are trained on corpora of multi-turn dialogues describing real programming tasks.

Yet obtaining such *domain-specific* dialogue data remains difficult. Public corpora are rich in data pertaining to popular ecosystems, but lack data about niche or private codebases. *Retrieval-augmented generation* (RAG) can help LLMs overcome this knowledge gap, but conversational data is still important to shape the behavior of a model: style, safety rules, and how to integrate retrieved snippets. Fine-tuning a model on relevant training data can also increase accuracy in retrieval tasks beyond RAG alone, particularly in smaller models [3].

Therefore, training an assistant to specialize in an unfamiliar domain requires a method for collecting domain-specific training dialogues. Creating fresh conversations by hiring experts is slow and expensive, so companies and researchers increasingly turn to *synthetic data* generation: prompting an LLM to talk to itself, then fine-tuning another model on the results. The *Self-Instruct* framework popularized this idea for general instruction following [4], and recent surveys list dozens of LLM-generated datasets across text and code [5].

However, naïvely sampling "self-chats" suffers from a number of short-comings. One concern is *coverage complexity*: artificially-generated samples should be made to avoid drifting towards "happy-path" scenarios and ensure domain coverage by visiting edge cases such as ambiguous queries or repeated failures [6], [7]. Another challenge is *domain grounding*, as LLMs frequently hallucinate API elements or usage details that can corrupt downstream training [8]. Additionally, free-form generations offer no latent structure for debugging or content-filtering, leading to poor *audibility* [9].

Beyond issues of output-quality, organizations often face strict privacy constraints that rule out sending proprietary code to external AI services, and the price of invoking frontier-grade models quickly becomes prohibitive when scaling to thousands or millions of synthetic dialogues. To enable efficient development of conversational assistants for software engineering tasks, we seek an *efficient* data-generation method that is controllable, reliable, and auditable.

In this paper, we present a pipeline to generate *structured* synthetic conversational data for the API search domain. We repurpose a *dialogue manager* (DM) for interactive API search as a *planner*. The DM emits a conversational "script" grounded in a target API comprising a sequence of user- and system- dialogue acts (DAs), e.g., `Provide-Query`, `Suggest`, `Elicit-Info`. A modern LLM realizes each act as a fluent, natural-language utterance, and a lightweight model is then distilled from the synthetic chats. The planner guarantees structure; the LLM supplies natural language. By decoupling symbolic planning from neural realization, the pipeline mass-produces domain-grounded dialogues at low cost while exposing every step as an inspectable trace.

We demonstrate this pipeline in a tool, *APIDA-Chat* (API Dialogue Act Chat), using publicly available services and data. There are 3 components, as illustrated in Figure 1:

1) **Dialogue Planner** – a 2021 DM for API search [10] emits dialogue act scripts referencing real API symbols.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00382

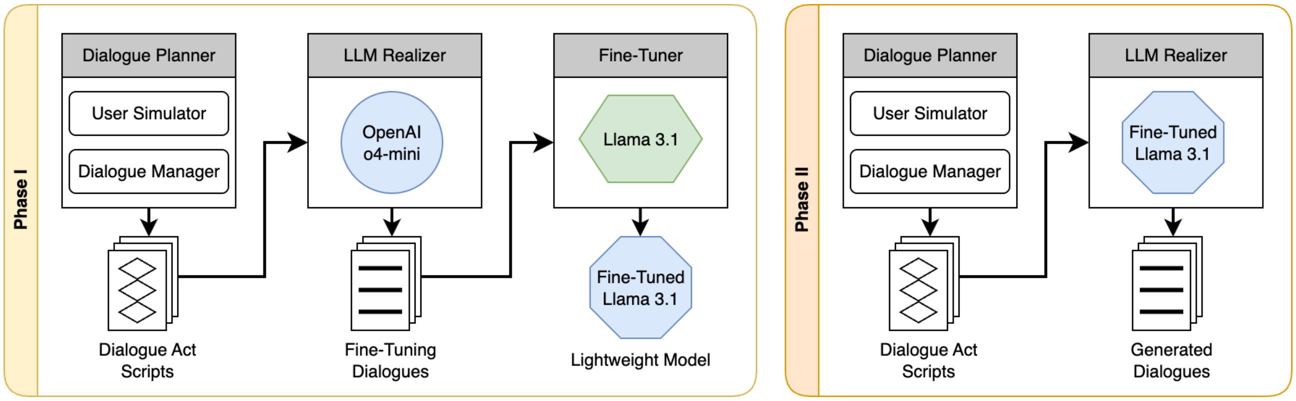

Fig. 1. Two-phase pipelines. Phase I bootstraps a lightweight model; Phase II reuses the planner with the fine-tuned model for low-cost dialogue generation.

2) **LLM Realizer** – a high-quality LLM consumes a script and produces a multi-turn chat realizing every act.
3) **Fine-Tuner** – a LoRA adapter trains a lightweight model on a small corpus of high-quality synthetic dialogues.

The fine-tuned model can then be used as the LLM realizer in conjunction with the dialogue planner to generate synthetic dialogues on controlled hardware, at a fraction of the cost.

Our reference implementation relies on deliberately lightweight components, but guarantees act-level coverage and domain grounding that naïve self-chat lacks. Stronger planners or retrieval-augmented generators are drop-in replacements, so our demonstration establishes a conservative lower bound.

We contribute: (1) a **planner-guided data generator** for bootstrapping API search chatbots; (2) an **open corpus and code release**, including 300 script-chat pairs for the Allegro C API; and (3) early **empirical evidence** that our fine-tuning pipeline enables a lightweight model to generate dialogues comparable to those produced by an advanced-reasoning model. This tool aims to serve researchers and enterprise teams looking to generate conversational training data for niche or private APIs, as well as those with rule-based API search tools who want to integrate LLM fluidity.

## II. Pipeline Overview

The pipeline in Figure 1 shows how *APIDA-Chat* turns a handful of API-aware "scripts" into an unlimited supply of domain-grounded training data. *Phase I* bootstraps a compact model using "teacher-student" pattern, and *Phase II* drops the costly teacher, using the fine-tuned student model to generate new dialogues at a fraction of the compute cost. Each component in our open-source tool is modular, and can be swapped for stronger/customized tools without modifying the rest of the pipeline.

### A. Phase I – Bootstrapping a Fine-Tuned Model

In Phrase I, a dialogue planner writes a symbolic script, a frontier-grade "teacher" LLM realizes the scripts into natural language conversations, and a LoRA fine-tuner distills those dialogues into a lightweight "student" model.

*1) Dialogue Planner:* The purpose of the dialogue planner is to generate a sequence of dialogue acts that might be observed in a real conversation in which the assistant efficiently guides the user to their goal. A dialogue act (DA) is a high-level abstraction of a conversational turn comprising a dialogue act type and relevant variables – e.g., `Provide-Query("bitmaps four needed dynamically")`. By emitting acts rather than surface text, the planner helps guarantee intent coverage, enforce safety rules, and determine concrete API entities for downstream components.

We use a dialogue planner for a conversational API search system published by Eberhart et al. [10]. The planner is split into a *user simulator* that follows heuristic rules derived from a Wizard-of-Oz study and a *dialogue manager* trained via deep Q-learning. The key idea is to model a user who exhibits realistic behavior, and an assistant that follows an optimal policy to help the user with their task. At each turn, the simulator chooses a user act based on conversation state and stochastic behavior parameters. The DM observes that act, queries a TF–IDF retriever over a knowledge base of API documentation, and outputs a system act. We demonstrate the tool with a dialogue policy trained using 5 million self-play steps and a turn-penalty reward (details unchanged from [10]).

*2) LLM Realizer:* Next, we use an advanced-reasoning LLM to generate a small, high-quality dataset of natural-language dialogues. These dialogues will be used to fine-tune our final model, so it is important to use an LLM capable of generating realistically expressive and varied search scenarios and dialogue turns. We adopt OpenAI's `o4-mini` (Apr 2025 snapshot) as the teacher LLM because it delivers strong code-reasoning and fluent dialogue at a reasonable cost.

We embed the full DA script in a single prompt and ask o4-mini to produce the whole dialogue in one pass. One-shot realization is roughly six times lighter on context tokens than alternating user/assistant calls, and it can yield more coherent exchanges because the model can plan across turns. Prior work on shopping agents [11] and interview generation [12] report mixed results comparing one-shot and multi-step prompting, so we adopt the cheaper option by default.

The prompt itself has two blocks. The *system* message frames the task and packs several practical rules – invent a plausible project context, weave every supplied keyword into realistic code or prose, open with a bridge sentence when the next act diverges from the user's last remark, vary tone and length on both sides, keep cross-turn references coherent, and never reveal API symbols unless the script calls for them. The subsequent *user* message embeds concise DA definitions, reiterates style constraints, and finally appends the DA script itself. The model must return a single JSON array of realized turns. These instructions were refined over multiple ad-hoc prompt iterations; they live in a separate text file so practitioners can adjust wording or add domain-specific constraints without touching code.

*3) Fine-Tuner:* This step distills the teacher-generated conversations into a compact student model that fits on a single consumer GPU. We start from a `Llama-3.2-3B-Instruct` model: an instruction-tuned model whose 4-bit quantization keeps memory demands low. We fine-tune the model by training a lightweight LoRA adapter over the synthetic corpus with a modest learning rate, leaving the original weights unchanged.

### *B. Phase II – Production-Scale Dialogue Synthesis*

In Phase II, the same dialogue planner produces fresh DA scripts, but now the fine-tuned student realizes each script locally. Users may loop this phase indefinitely, schedule nightly runs on new APIs, or plug the student into an online system: the pipeline's earlier stages remain unchanged, and higher-capacity planners or retrieval components can be swapped in.

### *C. Limitations*

The current prototype inherits several constraints from its lightweight design. First, the legacy dialogue planner used was trained on a single pre-processed API with a TF-IDF retriever, so its action space and grounding quality are narrow and fairly brittle. Second, the teacher realizer answers from parametric knowledge only, and is not immune to hallucination. Third, our prompt template is hand-tuned and may need adaptation in other domains, while the seed set of 250 dialogues risks style bias and gaps in edge-case coverage. Finally, the pipeline still depends on a proprietary cloud model in Phase I, which some privacy-conscious organizations cannot use.

These weaknesses are largely modular. A newer or custom planner can drop into the existing interface, and a retrieval-augmented realizer would help curb hallucinations. Because prompts live in a standalone text file, prompt tuning is straightforward. Phase I can be looped to enlarge the synthetic corpus, optionally mixing in selective human review. Lastly, teams that require full on-prem control can switch the teacher to an open 70 B model. Taken together, these upgrade paths suggest that our results should be viewed as a conservative baseline rather than an upper bound.

TABLE I
DIALOGUE LENGTH, DIVERSITY, AND SIMILARITY TO TEACHER.

| Model | Avg. Len | # Unique | BLEU | BERTScore |
|---|---|---|---|---|
| o4-mini | 99.96 | 6925 | N/A | N/A |
| llama | 82.58 | 3147 | .381 | .881 |
| llama-ft | 104.88 | 5138 | .497 | .906 |

## III. SYNTHETIC DIALOGUE DATASET

To illustrate the full pipeline, we demonstrate the *APIDA-Chat* tool targeting the `Allegro 5.0` C multimedia library – the same domain used in the original dialogue-manager study. Phase I automatically generates **250** DA scripts, realizes them with o4-mini, and fine-tunes the Llama-3.2 3B student. Phase II then produces an additional **50** held-out scripts that are realized by all three models (teacher, base student, fine-tuned student) for comparison.

### *A. Dataset*

The resulting training set contains 250 multi-turn conversations (≈180 k tokens). Table I summarises basic statistics: the fine-tuned student matches the teacher's average turn length (105 vs 100 tokens) and nearly doubles the vocabulary size of the untuned student, indicating richer lexical diversity. Against the teacher references, BLEU rises from 0.38 to 0.50 and BERTScore from 0.88 to 0.91 after fine-tuning. These intrinsic scores measure similarity to the teacher, not end-user quality; a fuller evaluation with human or automated preference judgments is left to future work.

### *B. Examples*

To give a small glimpse into the behavior of the three models, Figure 2 juxtaposes the first four turns of the *same* dialogue-act script realized by each model. In the first turn, the user provides a query; in the second, the AI assistant suggests a relevant function; in the third, the user requests information about the function; and in the fourth, the assistant attempts to answer the user's question.

The teacher model (o4-mini) transforms the sparse keyword list – "distinct, 5000, holes, fixasin" – into a plausible scenario ("triangulate a shape with 5000 distinct holes...") and then weaves that scenario into subsequent turns. The fine-tuned student (llama-ft) mirrors this structure but betrays its smaller capacity: it misinterprets fixasin as a verb ("fixasin a 5000 hole polygon"), yet still respects the act sequence and maintains cross-turn coherence. The baseline model (llama) illustrates the importance of the synthetic corpus: it drops keywords, gives a generic suggestion that ignores the user's context, and offers a vague description of the `al_fixasin` function that lacks salient details (signature, range, allocation safety) provided by the other two models.

In short, fine-tuning on just a few hundred planner-guided dialogues closes much of the gap to the teacher, while the zero-shot model frequently falls short on both grounding and conversational flow.

### C. Reproduction & Access

*APIDA-Chat* is publicly available under at https://github.com/Zeberhart/apida-chat. The repository bundles command-line scripts for each pipeline component together with a notebook that invokes them sequentially. Running the notebook on a standard Colab T4 instance reproduces all artifacts reported in this paper without code modification.

`Turn 1 (User)`

| | |
|---|---|
| **DA** | *Provide-Query(”distinct 5000 holes fixasin”)* |
| **o4-mini** | I'm trying to triangulate a shape with 5000 distinct holes, but my custom fixasin logic keeps failing—any ideas? |
| **llama** | I'm trying to create a 5000-hole fixasin, but I'm not sure what I'm doing—can you help me out? |
| **llama-ft** | I'm trying to fixasin a 5000 hole polygon in my 3D model—any ideas on how to handle that distinct geometry? |

`Turn 2 (System)`

| | |
|---|---|
| **DA** | *Suggest(al_fixasin)* |
| **o4-mini** | You might try al_fixasin(); it computes a fixed-point arcsin robustly, which could stabilize your angle math in those hole vertices. |
| **llama** | Have you considered using 'al_fixasin'? It's a function that can help you fixasin a given number of holes. |
| **llama-ft** | You might try al_fixasin() to convert your angle to a fixed-point representation, which can help with precision in your 3D math. |

`Turn 3 (User)`

| | |
|---|---|
| **DA** | *Elicit-Info(al_fixasin)* |
| **o4-mini** | Can you give me the details on al_fixasin? |
| **llama** | What exactly does 'al_fixasin' do? |
| **llama-ft** | Can you give me the details on al_fixasin? What input and output does it take? |

`Turn 4 (System)`

| | |
|---|---|
| **DA** | *Info(al_fixasin)* |
| **o4-mini** | al_fixasin(AL_FIXED x) returns $\sin^{-1}(x)$ as a 16.16 fixed-point value; it clamps inputs outside [-1,1] and avoids floating-point. |
| **llama** | 'al_fixasin' is a function that takes an integer as input and returns a fixed-point value. It's often used in graphics and game development to fixasin holes in a shape. |
| **llama-ft** | al_fixasin(AL_FIXED y) returns the inverse sine of y in fixed-point format (16.16). It's zero-alloc and safe to use in tight loops. |

Fig. 2. Example turns generated by different models for a given DA script.

## IV. CONCLUSION

*APIDA-Chat* demonstrates that a lightweight dialogue planner, a high-capability teacher LLM, and a modest LoRA finetune are sufficient to bootstrap a domain-grounded conversational model for API search. By separating symbolic planning from realization, the pipeline delivers three practical advantages: (1) DA-level coverage and grounding, (2) inspectable generation traces for safety and debugging, and (3) a compact student that can run locally without per-call cloud fees.

The architecture is intentionally modular; stronger planners, retrieval-augmented realizers, or alternative teacher/student models can be swapped in with minimal effort. Future work will explore improved RAG, automatic prompt search, and human-in-the-loop filtering to further improve coverage and factual accuracy. We hope the tool and dataset released with this paper will serve as a reproducible starting point for specialized, controllable dialogue data generation in the software-engineering domain.

## REFERENCES

[1] S. Overflow, “Stack Overflow Developer Survey 2024,” https://survey.stackoverflow.co/2024/, 2024, accessed: July 12, 2025.

[2] A. Sergeyuk, Y. Golubev, T. Bryksin, and I. Ahmed, “Using ai-based coding assistants in practice: State of affairs, perceptions, and ways forward,” *Information and Software Technology*, vol. 178, p. 107610, Feb. 2025. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2024.107610

[3] H. Soudani, E. Kanoulas, and F. Hasibi, “Fine tuning vs. retrieval augmented generation for less popular knowledge,” in *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, ser. SIGIR-AP 2024. ACM, Dec. 2024, p. 12–22. [Online]. Available: http://dx.doi.org/10.1145/3673791.3698415

[4] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, “Self-instruct: Aligning language models with self-generated instructions,” 2023. [Online]. Available: https://arxiv.org/abs/2212.10560

[5] M. Nadăș, L. Dioșan, and A. Tomescu, “Synthetic data generation using large language models: Advances in text and code,” *IEEE Access*, p. 1–1, 2025. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2025.3589503

[6] W. Cui and Q. Wang, “Ada-instruct: Adapting instruction generators for complex reasoning,” 2024. [Online]. Available: https://arxiv.org/abs/2310.04484

[7] S. Steindl, U. Schäfer, and B. Ludwig, “CoPrUS: Consistency preserving utterance synthesis towards more realistic benchmark dialogues,” in *Proceedings of the 31st International Conference on Computational Linguistics*, O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, and S. Schockaert, Eds. Abu Dhabi, UAE: Association for Computational Linguistics, Jan. 2025, pp. 5902–5917. [Online]. Available: https://aclanthology.org/2025.coling-main.394/

[8] F. Liu, Y. Liu, L. Shi, H. Huang, R. Wang, Z. Yang, L. Zhang, Z. Li, and Y. Ma, “Exploring and evaluating hallucinations in llm-powered code generation,” *arXiv preprint arXiv:2404.00971*, 2024.

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[10] Z. Eberhart and C. McMillan, “Dialogue management for interactive api search,” 2021. [Online]. Available: https://arxiv.org/abs/2107.12317

[11] X. Li, Z. Chen, J. I. Choi, N. Vedula, B. Fetahu, O. Rokhlenko, and S. Malmasi, “Wizard of shopping: Target-oriented e-commerce dialogue generation with decision tree branching,” 2025. [Online]. Available: https://arxiv.org/abs/2502.00969

[12] J. D. Baer, A. S. Doğruöz, T. Demeester, and C. Develder, “Single- vs. dual-prompt dialogue generation with llms for job interviews in human resources,” 2025. [Online]. Available: https://arxiv.org/abs/2502.18650