ASPERA: A Simulated Environment to Evaluate Planning for Complex Action Execution

Alexandru Coca[1*], Mark Gaynor[2], Zhenxing Zhang[2], Jianpeng Cheng[3*], Bo-Hsiang Tseng[2], Pete Boothroyd[2], Héctor Martinez Alonso[2], Diarmuid Ó'Séaghdha[2], Anders Johannsen[2]

[1]Department of Engineering, University of Cambridge, [2]Apple, [3]Meta
ac2123@cam.ac.uk, ajohannsen@apple.com

Abstract

This work evaluates the potential of large language models (LLMs) to power digital assistants capable of complex action execution. These assistants rely on pre-trained programming knowledge to execute multi-step goals by composing objects and functions defined in assistant libraries into action execution programs. To achieve this, we develop ASPERA, a framework comprising an assistant library simulation and a human-assisted LLM data generation engine. Our engine allows developers to guide LLM generation of high-quality tasks consisting of complex user queries, simulation state and corresponding validation programs, tackling data availability and evaluation robustness challenges. Alongside the framework we release *Asper-Bench*, an evaluation dataset of 250 challenging tasks generated using ASPERA, which we use to show that program generation grounded in custom assistant libraries is a significant challenge to LLMs compared to dependency-free code generation[1].

1 Introduction

Digital assistants, such as Siri and Alexa, provide a conversational interface for users to execute *simple actions* (e.g., *Set a timer for 5 minutes*). To achieve this, developers typically define APIs (intents) and collect data to train specialised parsing models responsible for translating user requests into machine-interpretable, domain-specific languages that can execute these APIs (Andreas et al., 2020; Cheng et al., 2020). Equivalently, action execution in this setting can be modelled as a function call to an intent API implemented by a target application (e.g., `alarm_set_timer(duration=5, unit='min')`) Function calling supports simple actions, but extension to execute *any* action on the device requires implementation of fine-grained intents and/or specialised parsing functions for an intractably large number of requests. To enable future digital assistants to execute *complex actions* (Figure 1), Jhamtani et al. (2024) propose generation of a program implemented with low-level *primitives* from assistant libraries[2]. We aim to evaluate the ability of LLMs to generate such programs when (1) the LLM has access to all the relevant information for generation, encoded in the assistant library documentation, or (2) the LLM selects the relevant primitives by exploring the entire assistant library as a first step prior to program generation. To this end, we address two challenges.

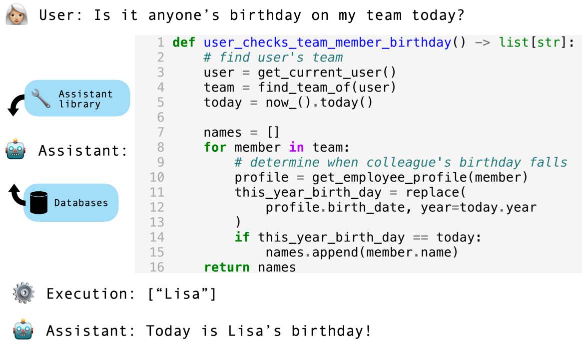


Figure 1: Example of a digital assistant executing a complex action given primitives (e.g., now_) defined in a custom assistant library and databases containing user's data. The assistant decomposes the query and calls 5 APIs (lines 3 - 5, 9 - 10), performing attribute access, passing values by attribute reference (line 11 - 13), in addition to iteration and flow control in a multi-step program to achieve the user's goal. Logical reasoning is required to deduce that the year of the birthday has to be updated to the current year.

1. Complex action evaluation instances comprising diverse, realistic queries annotated with programs requiring compositional use of multiple primitives are required for evaluation. Existing resources do not fully satisfy this requirement. SM-

[*]Work done while at Apple.

[1]https://github.com/apple/ml-aspera.

[2]The assistant library is a collection of functions and objects the assistant can use to compose plans which determine or change the user's device state. A *primitive* is any abstraction implemented in the library (e.g., a function or class).

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 25399–25434
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

CalFlow (Andreas et al., 2020) contains compositional queries but is annotated with a specialised domain-specific language (DSL) which hinders LLM performance (Bogin et al., 2024). DeCU (Jhamtani et al., 2024) is a dataset for evaluating plan generation for complex user queries, but evaluates solutions using an LLM judge rather than implementing an executable simulation of the assistant library. In lieu of documentation—which has been shown by recent research to improve LLM performance on many tasks (Lu et al., 2024; Srivastava et al., 2024; Hsieh et al., 2023)—DeCU provides only in-context examples (ICEs) primarily demonstrating how to parse simple user queries into single-instruction programs. Styles et al. (2024) and Trivedi et al. (2024) develop simulated environments with comprehensively documented APIs, but limit action diversity by grounding queries in task templates. Unlike the approach proposed in this paper, environment simulation and functional correctness validation in these environments relies on human effort and expertise alone.

2. Robust evaluation of complex action execution capability requires measuring task success, i.e. that the assistant actions satisfy the user goal. Jhamtani et al. (2024) note this to be an open problem, since functional correctness evaluation requires query-dependent databases and accounting for unwarranted side-effects[3]. Styles et al. (2024) tackle this by feeding databases to templated executable programs to annotate expected environment states. They propose strict database comparisons to estimate task success, and hence cannot evaluate queries with multiple outcomes and *information-seeking queries*[4]. Trivedi et al. (2024) address these limitations, but define environment states and evaluate task success via specialised programs implemented by domain experts for every task.

Contributions We propose ASPERA, a simulated environment supporting evaluation of agents capable of complex action execution with data generation capability. Given an assistant library simulation (§2.1), ASPERA enables a developer and an LLM to interact to generate diverse, high-quality complex user requests and programs which satisfy them. We show that robust task success estimation is possible for both synthesised and human-authored queries by prompting LLMs to generate

[3]This term describes an unintended action by the agent e.g., setting a meeting with the wrong attendees.

[4]Queries where information is provided to the user.

Module	Functions	Classes	Docs length (words)
time utils	22	11	986
work calendar	13	3	660
company directory	10	3	236
room booking	4	2	331
exceptions	-	1	209
Total	49	20	2,422

Table 1: Assistant library summary statistics. A module corresponds to a `.py` file. Docs length is the total length of the documentation strings defined inside the module. See App. B,`src/aspera/apps_implementation` and `src/aspera/apps` in our code release for details.

programs which appropriately initialise the environment state and determine whether the executed action satisfies the user goal (§2.3.2 and 2.3.3). Using this system, we address the lack of complex actions execution data by generating *Asper-Bench*, a challenging collection of 250 tasks (§3). Evaluation on this dataset shows that (1) generating programs that satisfy complex action requests is a challenge for LLMs even when they are prompted with all the relevant information, despite their ability to generate plausible programs and (2) state-of-the-art (SOTA) LLMs find it difficult to select all the primitives needed for composite tasks, adding a challenge to program generation (§5 and 6).

2 The ASPERA Framework

In ASPERA, a human developer initiates an interactive session in which an LLM is prompted to generate complex user requests grounded in a `python` library which can implement digital assistant use cases. In subsequent human-LLM interactions, two additional programs which enable success rate evaluation for arbitrary agents are generated. We now discuss how this works in practice.

2.1 Assistant library

ASPERA implements an assistant library which simulates a company in which employees in various teams (with a tree-based reporting structure) have meetings with one another under various conditions, managed by a room booking system. The library consists of 7 databases and 69 `python` primitives (Table 1). An extensive time utilities library, partially inspired by SMCalFlow (Andreas et al., 2020), is implemented to test logical and arithmetic reasoning capabilities.

2.2 Components of an ASPERA task

A task generated by ASPERA has four elements: (1) the *user query*, a natural-language request for the assistant to execute an action (e.g., *Cancel my*

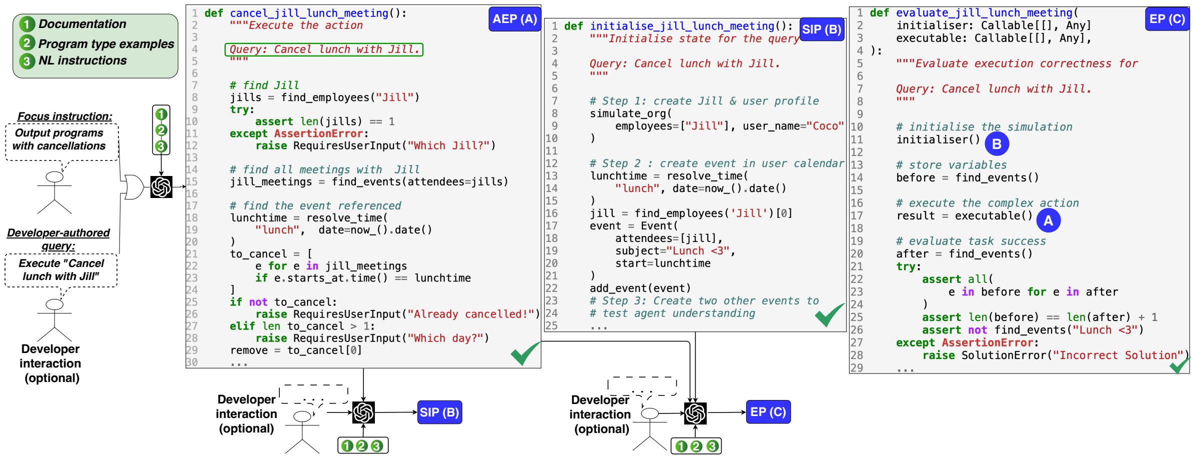


Figure 2: Sample ASPERA task, depicting action execution (A), state initialisation (B) and evaluation (C) programs. The task is generated in an interactive chat session (§2.4) which is initialised with AEP generation prompts (App. A.1 or A.2). To ground state initialisation, the chat history is extended with SIP generation prompt (App. A.3), which developers can customise with task-specific instructions. Finally, the chat history is extended with the EP generation prompt (App. A.4) which can also be customised by developers via instructions. At each step, the developer can execute and edit the generated programs to ensure data quality.

lunch with Jill); (2) the *action execution program* (AEP), a program which satisfies the user request upon execution; (3) the *state initialisation program* (SIP), which uses the assistant library and simulation tools to set the environment state so that the query can be executed in `python` (i.e., establishing the existence of an employee named Jill and some meetings scheduled with her); (4) the *evaluation program* (EP) which runs the AEP in the initialised environment and determines its correctness (i.e., checks that the correct meeting has been deleted). Figure 2 depicts a simple ASPERA task.

2.3 ASPERA task generation

Figure 2 shows that the three programs which comprise a task are generated given: (1) assistant library documentation; (2) ICEs demonstrating the program format; and (3) natural language instructions. The instructions describe the assistant policy, environment assumptions and/or program structure information (depending on the program type to be generated).

2.3.1 Query and AEP generation

The user query can be authored by the human developer or synthesised by the LLM with the AEP (as part of the AEP docstring – see prompts in App. A.1 and A.2). By prompting the LLM with the documentation of the assistant library and with suitable examples, diverse and complex AEPs are generated. The complexity of the generated AEPs is characterised by: (1) number of primitives; (2) a variety of compositional patterns (Figure 2 AEP, lines 8 & 15, 18 - 20); (3) flow control and iteration (lines 21 - 24) and; (4) complex date-time reasoning (l. 18 - 20). Moreover, by prompting the LLM with exceptions, the AEPs model disambiguation (lines 9 - 12, 27 - 28) and unsatisfiable requests (lines 25 - 26). The AEP examples contain *planning steps*, that outline a possible decomposition of the task (lines 7, 14, 17) to encourage step-by-step thinking and to improve generation quality.

Bias mitigation Since ASPERA relies on LLMs for data generation, biases inherent to these models may propagate into the generated datasets, potentially limiting task diversity. To address this issue, we employ three techniques. First, inspired by Wang et al. (2024a), we append the history of previously generated queries to the AEP generation prompt, explicitly instructing the model to create novel tasks and thereby reduce repetition bias. Second, we condition task generation on interactively specified *focus instructions*, which developers can use to control query attributes such as complexity, length, or scenario context[5]. This interactive specification allows direct influence on data diversity (see Figure 7 in App. A.1). Third, queries can be interactively filtered post generation, providing an additional human-driven mechanism to mitigate biases before dataset finalisation (§2.4).

[5]For instance, a focus instruction might be: *"Generate queries requiring coordination of schedules among organisation members."*

2.3.2 SIP generation

After AEP generation, the LLM is prompted (see App. A.3) to generate an SIP, which initialises the simulation so that the outcomes of an agent's actions can be evaluated. The SIP re-uses the primitives implemented for action execution (Figure 2 SIP, lines 13 - 22). This obviates the need for handcrafting databases, using templates to define the user query or prompting the LLM with database schemata. While statically defined databases model a single user's behaviour, ASPERA's dynamic database generation allows it to model multiple users. To simplify generation of complex environment states (e.g., an organisation reporting structure) the LLM can call ASPERA simulation tools (lines 8 - 10; see App. A.5).

2.3.3 EP generation

The final step is to generate an EP, which enables ASPERA to evaluate the functional correctness of an AEP (see App. A.4). The EP takes as positional arguments the reference SIP and the AEP (Figure 2 EP, lines 2 - 4) and executes them in this order (lines 11 & 17) to initialise the environment and execute the user action. Prior to action execution, one or more variables (line 14) store the initial state relevant to assessing side-effects and user goal completion. After AEP execution, the variables are compared with their expected values in assertion statements (lines 22 - 26). These verify the user goal was met without unexpected side effects.

The EPs thus implement goal-oriented evaluation even though the environment state is implicit in the queries and SIPs. They generalise database comparison functions implemented in other environments (Lu et al., 2024; Styles et al., 2024) because they can evaluate information-seeking queries by comparing the AEP returned value against its expected value. Finally, evaluation of queries with multiple allowable outcomes[6] is supported in ASPERA by comparing captured state with a range of accepted values in assertion bodies.

2.4 Developer-LLM interaction in ASPERA

Figure 2 shows how AEP, SIP, and EP generation is sequential and moderated by a developer. As discussed in §2.3.1, the developer can seed the AEP generation with a focus instruction (top left) to ensure diversity or author the query and supervise AEP generation (bottom left). AEPs are generated as `python` scripts. Developers can add special directives above function signatures to filter low-quality or repetitive examples.

After AEP generation, the chat history is automatically extended with the SIP generation prompt. The developer can optionally instruct the LLM to customise the environment state to be generated, define multiple SIPs or implement new simulation tools the LLM can use to write the SIPs. The interactive loop is repeated to enable EP generation. At any point, the developer can execute the programs in the simulated environment and edit them (or the queries) accordingly to ensure data quality.

3 The *Asper-Bench* Dataset

We generate an evaluation dataset of 250 tasks using GPT-4o[7], given five ICEs for each program type (§2.2). 71 tasks are information-seeking, while the remainder mutate one or more databases. We include both LLM- and human-authored queries. A single SIP and EP are generated for each query, except for conditional queries (Table 2, line 7) where state initialisation and evaluation are defined to test each AEP branch. Our annotations contain 9k, 13k and 17.5k lines across AEPs, SIPs, and EPs.

Asper-Bench AEPs are diverse in their complexity (Figure 3). The distribution of maximum abstract syntax tree (AST) depth indicates AEPs satisfying the queries require compositional use of multiple primitives[8]; LLMs must interpret extensive documentation across multiple modules and demonstrate strong coding ability to generate AEPs which complete *Asper-Bench* tasks.

As further shown in App. C, the queries pose challenges ranging from parsing complex time expressions and date/time arithmetic (Table 2, rows 3, 8 - 10) to logical reasoning and interpretation of additional instructions (rows 3 - 5, see App. C.1). Hence, the dataset's diversity stems from task complexity, not paraphrasing.. Representing such complex queries as programs requires iteration and flow-control patterns. This increases a program's *cyclomatic complexity* (CC), defined as the number of independent paths that can be traversed during execution (McCabe, 1976). Tasks with higher CC involve non-trivial operations to resolve people, events or dates (Table 3, rows 8, 10), complex

[6]Multiple outcomes are defined for *When is Bob free next Friday?* since both the upcoming Friday or Friday the following week are valid interpretations of the date mentioned.

[7]`gpt-4o-2024-05-13`.

[8]For comparison, the maximum AST depth of an AEP containing a call where all slot values are strings (e.g., `find_events(subject="Paper Review")` is 5.)

Id	Query	Length (words)	Cyclomatic complexity	# primitives	Max. AST depth
1	Assistant, schedule lunch with my entire team tomorrow at noon.	12	1	7	6
2	Assistant, schedule lunch with a different team member each day next week at 12:30 PM.	17	3	8	10
3	Assistant, add a 1-hr strategy review with the CFO and the COO one week from today at 2:30.	23	5	13	9
4	Assistant, check my boss' calendar Wednesday to Friday next week, can they meet?	18	7	6	11
5	Assistant, I need to know which of Bill or Bob is busiest next week so I can allocate work.	21	7	7	14
6	Assistant, reorganise my diary on the fifth so that the important meetings come first.	16	9	10	11
7	Assistant, cancel the second meeting with Alice tomorrow if she declined.	13	8	5	10
8	Assistant, when in August when everyone from finance is off?	12	10	7	11
9	Assistant, set up a status update meeting with my manager every last Friday of the month at 2 PM till the end of the year. Skip his holidays.	33	10	16	10
10	Assistant, edit the attendee list for our fortnightly team planning on Wednesdays at 1 PM to remove Jack and Amy and add the newest sales hire.	28	13	11	10

Table 2: *Asper-Bench* sample queries (see §3).

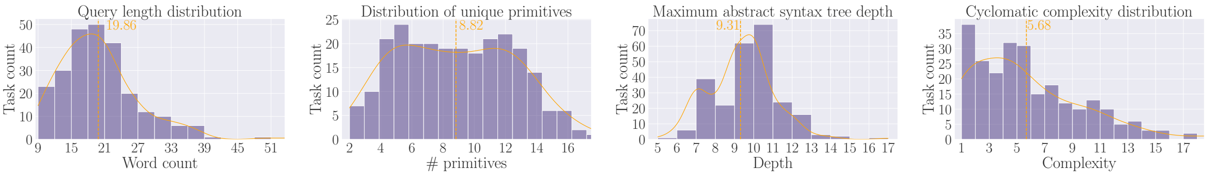


Figure 3: Distributions of key complexity measures in the *Asper-Bench* reference AEPs.

rescheduling (row 6) and scheduling events subject to constraints (row 9). Lower CC tasks test fine-grained documentation understanding and programming ability (row 1); occasionally, these tasks require branching to follow instructions which provide relevant information about the environment that does not naturally fit in the documentation (row 3) or describe the assistant policy[9] (App. C.3).

Quality control The ASPERA data generation engine is integrated with the developer's IDE. Consequently, the lead author, who has deep expertise in digital assistants, executed the tasks and used syntax highlighting and auto-completion features to efficiently correct LLM output. Two annotators with similar expertise confirmed the data quality while carrying out the error analysis in §6.

4 ASPERA Evaluator

ASPERA provides an interface which enables arbitrary agents to execute AEPs and observe execution outcome. To support ongoing comparison of the baseline complex action execution capability of LLMs independent of the agent prompt, we provide two implementations of this interface.

1. Complete codebase knowledge (CCK) The agent prompt (Figure 19, App. D) contains the documentation for the entire assistant library (Table 1) alongside the five AEP example used to generate *Asper-Bench*. The prompt also includes instructions for: an events scheduling policy; information about environment constraints[10]; and the output format. For information-seeking queries, the type of the object to be returned to the caller is also included in the prompt.

2. Primitives selection (PS) The primitives are not known when the user invokes the assistant. Including the entire assistant library documentation in the prompt (as in the CCK prompt) may be impractical due to context window and latency limitations. In such a case, the assistant must inspect the library to determine which primitives are needed to execute the action requested by the user. To evaluate how well agents perform under these constraints, we provide a simple interface in which AEP generation is conditioned on primitives selected by the LLM prior to generation. This involves an iteration through an extended assistant library[11]. At each step, the agent is prompted with the documentation for an ASPERA module (viz Table 1) alongside the user request and is asked to issue `import` statements to select relevant primitives or None if the module is not relevant for executing the requested action (Figure 20a, App. D). On iteration completion, the selected primitives replace the full application library listings in the CCK prompt.

Unlike the CCK prompt, which includes 5 ICEs, the PS AEP generation prompt contains only one example demonstrating the solution format. Including the CCK examples would have inflated success rates for agents with poor primitive selection recall, as the ICE primitives and their documentation would appear without being explicitly imported.

[9] For details, see Figure 8 in App. A.1.

[10] These include e.g. company information (e.g., *The leadership team is formed of a CEO, COO and CFO.*).

[11] The extension contains documentation for the `ai_assistant`, `contacts`, `files`, `messaging`, `navigation`, `user_device_settings` modules in addition to those reported in Table 1, to be implemented in a future release.

Model name	Checkpoint	Size	Task success (%)	Syntax err. (%)
o1	o1-preview-2024-09-12	-	80.13	-
o1-mini	o1-mini-2024-09-12	-	51.40	0.13
GPT-4o	gpt-4o-2024-05-13	-	45.33	-
GPT-4o-mini	gpt-4o-mini-2024-07-18	-	21.07	-
3.5-turbo	gpt-3.5-turbo-0125	-	10.80	1.20
1.5-pro	gemini-1.5-pro-002	-	33.73	0.40
1.5-flash	gemini-1.5-flash-002	-	27.87	0.40
1.0-pro	gemini-1.0-pro-002	-	12.67	0.53
Mistral L	Mistral-Large-Instruct-2407	123B	38.00	-
Qwen2.5	Qwen2.5-72B-Instruct	72B	28.80	-
Gemma2	gemma-2-27b-it	27B	14.40	0.4
CodeGemma	codegemma-7b-it	7B	2.40	6.0

Table 3: CCK *Asper-Bench* task completion rates (5-shot). Proprietary model results are averaged over three runs. Greedy decoding is used for all models except o1, which only supports a temperature of 1. See App. F for further evaluations.

Model name	Setting	# ICE	Micro F1	P	R	Task success (%)
o1	CCK	5	-	-	-	80.13
	CCK	1	-	-	-	72.80
	PS	1	0.63	0.60	0.67	28.40
GPT-4o	CCK	5	-	-	-	45.33
	CCK	1	-	-	-	36.53
	PS	1	0.56	0.56	0.55	11.46

Table 4: PS task success. Rows 1 and 4 are repeated from Table 3, # ICE denotes the number of AEP examples in the prompt. Precision and recall are computed with respect to the *Asper-Bench* reference AEPs.

Metrics We report task success. A task is completed if the generated AEP executes without error and all assertions pass in all reference EPs.

5 *Asper-Bench* Evaluation

Complete assistant library knowledge (CCK setting) AEP generation is challenging for both proprietary and open-source LLMs even when they can directly observe all the knowledge relevant for planning (Table 3). Despite performing well on standard code generation benchmarks (Chen et al. (2021), Austin et al. (2021a)), and their ability to consistently generate syntactically correct AEPs (Table 3, column 5), the most widely used general-purpose assistants successfully execute only 45.33% (GPT-4o) and 33.73% (Gemini 1.5 Pro (Reid et al., 2024)) of actions. Task success correlates with model size (Table 3, r. 9-13). However, the improved task success of o1-mini compared to larger LLMs such as GPT-4o (+6.1%) and Gemini 1.5 Pro (+17.67%) suggests that scaling inference-time compute may be a key enabler of improved complex task understanding and execution capabilities.

Primitive selection (PS setting) Despite its AEP generation capability when conditioned on the documentation of the entire ASPERA library, o1 retrieves just 67% of the primitives relevant for AEP implementation and achieves a modest 28.4% task completion rate as a result (Table 4). Hence, while identifying which primitives are relevant for executing a given action is relatively simple for human developers, we find that SOTA LLMs have limited ability to perform in this setting.

Statistic	Model name				
	GPT-3.5-turbo	GPT-4o-mini	GPT-4o	o1-mini	o1
Lines of code Δ to reference AEPs	-12.15	-7.3	-5.48	3.22	8.72
`RequiresUserInput` usages	52	93	170	360	291
Average planning steps (viz. Figure 1, lines 2&9)	4.83	5.63	5.41	6.15	9.16
Helper functions count	0	2	11	29	65
Average cyclomatic complexity	2.92	3.82	4.44	5.95	6.80

Table 5: Key generated AEPs statistics

Model name	Programs debugged	Programs analysed	Errors labelled	Could recover (%)
GPT-4o	33	125	41	48.39
GPT-3.5-turbo	66	125	100	24.62

Table 6: Execution error analysis statistics.

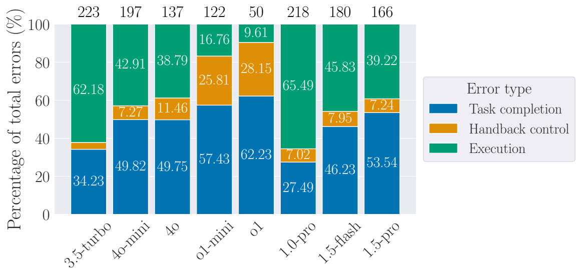


Figure 4: Assistant error types for OpenAI and Gemini model families. Top row displays total error counts.

6 Analysis and discussion

6.1 CCK error analysis

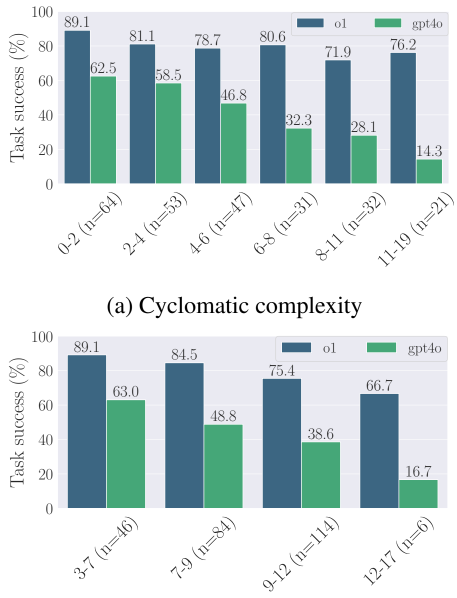
We begin with an in-depth analysis of programs generated by agents prompted with the documentation of the entire ASPERA library. A breakdown of the errors observed is presented in Figure 4. We make three key observations.

First, for both OpenAI and Gemini models, more capable variants produce a larger proportion of *task completion errors*, in which programs execute successfully but fail an assertion in evaluation. Such an error indicates that the model can successfully use and combine primitives, but fails to understand some nuance in the user request and therefore takes the wrong action. Table 14 (App. E.2) shows concrete examples of this.

Second, less capable models incur relatively more *execution errors*, in which programs are syntactically correct but trigger a runtime exception. An in-depth error analysis of 141 such errors from GPT-3.5-turbo and GPT-4o (App. E.1) shows that both models have a tendency to hallucinate in situations where multi-step reasoning is required, generating shorter AEPs compared to the reference annotations (Table 5, row 1). Additionally, we find

Subset	CC	AST depth	o1(%)	GPT-4o (%)	Example
Simple	1.9	7.3	100	100	Table 2, row 1
Constrained scheduling	7.1	9.6	86.67	46.67	Table 2, row 9
Complex time expressions	5.4	9.2	63.33	20.00	Table 2, r. 4 & 10
Policy / instruction following	6.0	9.2	80.00	20.00	Table 2, r. 2 & 3
Advanced problem solving	9.2	10.6	56.67	26.67	Table 2, row 5

Table 7: Task success for query subsets. Each subset has 10 queries, see App. E.4 for complete listings.



(a) Cyclomatic complexity

(b) Maximum AST depth

Figure 5: Task success as a function of reference AEP complexity (n denotes bucket size).

that execution errors often occur with task completion errors; in other words, the solution is incorrect even if the execution error is manually fixed (Table 6, column 5). While self-reflecting agents (Shinn et al., 2023) could achieve higher task success, our evaluation considers complex action execution in the single trial setting since, in practice, self-debugging iterations increase latency and trial execution might have unintended consequences.

Third, more capable models generate a greater proportion of *handback control errors*. These errors are linked to more frequent use of the `RequiresUserInput` exception (Table 5, row 2), used to handle cases in which the assistant cannot complete a task or cannot disambiguate between some entities at runtime. The errors occur when this exception triggers unexpectedly, indicating that the agent has made an incorrect assumption or misidentified an edge case. These errors illustrate which types of queries remain difficult for SOTA models (see Table 15, App. E.3).

6.2 Case study

Table 8 presents examples of the error categories introduced previously, highlighting the critical need for evaluation protocols that reflect real-world deployment conditions. While execution errors may be recoverable through repeated trials, task-completion and handback control errors are often irreversible, potentially resulting in misinformation to end-users. Crucially, as shown in Figure 4, these irreversible errors predominate in frontier models. Thus, reductions in these error types provide compelling evidence of enhanced reasoning and reliability in LLMs, underscoring the importance of targeted improvements in model evaluation.

6.3 Handling complexity

Asper-Bench requires models to perform various complex compositions of primitives and control flow sequences. Figure 5 shows that o1 can successfully complete a much larger proportion of tasks which require generating complex programs compared to GPT-4o. As seen in Table 5, o1 is more capable in this regard due to its ability to break down the task into fine-grained steps (row 3), make use of helper functions to encapsulate complex functionality (row 4) and to more effectively employ flow control and iteration (row 5).

To further demonstrate the challenges in *Asper-Bench*, we select 5 subsets of 10 queries which test different aspects of assistant understanding and reasoning capabilities. Table 7 (row 1) shows that both o1 and GPT-4o can equally handle simple problems (e.g., scheduling an event on a given date, or deleting events) but a large gap is observed in the completion rate of advanced tasks. Compared to o1, GPT-4o struggles with constrained scheduling and resolving challenging relative time expressions (rows 2 & 3), which require flow control, primitive composition and arithmetic reasoning. The same is true of generating AEPs constrained by additional instructions in the prompt (row 4) and solving very challenging examples from the above categories (row 5).

6.4 Primitives selection

The primitives selection setting proved challenging for both models evaluated, as shown in Table 4. The LLMs show limited ability to reason about dependent primitives. Using the `work_calendar` module, for example, requires knowledge about properties of the `Event` primitive. We find this relation is not recognised during selection; o1 fails to import both the relevant `work_calendar` API and `Event` in 29 out of 67 occurrences of `find_events`, 16 out of 69 occurrences of `add_event` and 8 out

<table>
<tr><th>Id</th><th>Query</th><th>Error Snippet</th></tr>
<tr><td>1</td><td>Assistant, schedule our team Christmas party 10 days before Christmas. Should start in the morning and end at 10 PM.</td><td><pre>
1 def schedule_team_christmas_party():
2
3   # find the user's team to determine event attendees
4   ...
5
6   # resolve the date for 10 days before Christmas
7   christmas_day = parse_date_string(DateExpressions.ChristmasDay)
8   party_date = modify(
9   christmas_day,
10   Duration(10, TimeUnits.Days),
11   operator=DateTimeClauseOperators.subtract,
12   ).date()
</pre></td></tr>
<tr><td colspan="3">Tool use (datetime): Line 9 contains a <code>TypeError</code>, <code>modify</code> only accepts <code>datetime</code> objects. A correct solution requires an additional reasoning step: pass <code>christmas_day</code> and one of the specified times to the <code>combine</code> library function to get the correct type.</td></tr>
<tr><td>2</td><td>Assistant, set up a training session for all employees from the Engineering team next Monday from 2 PM to 5 PM. Send out invites and book a conference room that fits 20 people.</td><td><pre>
1 def schedule_engineering_training_session():
2
3   # Find all employees in the Engineering team
4   engineering_team = [
5   emp for emp in get_all_employees()
6   if emp.team == Team.Engineering
7   ]
</pre></td></tr>
<tr><td colspan="3">Attribute hallucination: In line 6, the <code>.team</code> attribute access raises an error because the <code>Employee</code> objects returned by <code>get_all_employees</code> only have <code>name</code> as an attribute. The <code>Employee</code> object should be passed instead to the <code>get_employee_profile</code> library function to return an object which has <code>team</code> as an attribute.</td></tr>
<tr><td>3</td><td>Assistant, put 45 minutes in the calendar, back-to-back, with Engineering and Marketing starting at 10 AM tomorrow... Actually, add a 10-minute buffer between each meeting.</td><td><pre>
1 def schedule_back_to_back_meetings():
2
3   # find the user's profile
4   ...
5
6   # find the teams
7   engineering_team = find_team_of(Employee(name="Engineering"))
8   marketing_team = find_team_of(Employee(name="Marketing"))
</pre></td></tr>
<tr><td colspan="3">No tool use (lazy solution): The assistant hallucinates lines <code>7-8</code> instead of using relevant APIs to find the engineering team, in spite of documentation that states that <code>Employee</code> objects cannot be instantiated. The functions <code>get_all_employees</code>, <code>get_employee_profile</code> and the enumeration <code>Team.Engineering</code> should have been composed, similar to snippet in row 2.</td></tr>
</table>

(a) Execution errors examples.

Id	Query	Agent action
1	Assistant, Ari and James are on holiday next month, who's out for longer?	Sums duration of all vacations, month notwithstanding.
2	Assistant, add bi-weekly mentorship sessions with the reports of my reports starting next Monday at 2 PM to my calendar.	Hallucinates an end date for the event, scheduling instances only for 6 months.
3	Assistant, reschedule the meetings which overlap with the annual review this afternoon to the same time tomorrow.	Creates copies of overlapping events tomorrow, instead of modifying existing events.

(b) Task completion errors examples.

<table>
<tr><th>Id</th><th>Query</th><th>Agent action</th></tr>
<tr><td>1</td><td>Assistant, find a suitable conference room for a meeting with my team I wanna schedule later today.</td><td>Tries to schedule a meeting, handing back control because of incorrect diary checking.</td></tr>
<tr><td colspan="3">Error cause: Distracted by irrelevant information. The agent is not required to schedule a meeting – not enough details are provided. It should search for an available room that has sufficient capacity to accommodate the user and their team instead.</td></tr>
<tr><td>2</td><td>Assistant, can you find a time slot in my diary today when I could schedule something with the HR department to discuss my performance review?</td><td>Hallucinates a program attempting to find HR team, handing back control because it cannot determine it.</td></tr>
<tr><td colspan="3">Error cause: Distracted by irrelevant information. The HR team is not defined in the simulation. The task requires the agent to find a slot in user's diary.</td></tr>
<tr><td>3</td><td>Assistant, schedule our team Christmas party 10 days before Christmas. Should start in the morning and end at 10 PM?</td><td>Requires the user to provide an alternative date.</td></tr>
<tr><td colspan="3">Error cause: Following policy. The agent follows the instruction Unless the user explicitly states the date, meetings should not be scheduled on or recur during weekends. which is irrelevant.</td></tr>
</table>

(c) Handback control errors examples.

Table 8: Error examples (CCK setting). Further examples are provided in Tables 13 to 15 in App. E.

of 19 occurrences of `delete_event`. The ability of an LLM to use a primitive listed in the prompt is weakly associated with its selection performance for that primitive. In our baseline setting (CCK, 1-shot), o1 achieves a 66% task success rate on queries where the reference AEP uses `add_event`. However, its recall for selecting `add_event` is just 0.41, with an F1 score of 0.58 (see App. E.5). This suggests that selecting a complete set of fine-grained primitives to execute complex user requests is challenging for LLMs.

7 Related Work

Task-oriented parsing Parsing natural language queries into DSL programs interpretable by execution engines (Zelle and Mooney, 1996; Gupta et al., 2018, *inter alia*) is challenging for program structures unseen in training (Yao and Koller, 2022). Bogin et al. (2024) and Jhamtani et al. (2024) show that representing targets as *programming languages* improves LLMs' few-shot semantic parsing ability; we build on this, employing program synthesis to collect complex, high-quality, task-oriented queries and to evaluate agents' ability to understand them.

Tool-augmented LLMs An alternative is query synthesis at scale by prompting LLMs with documentation of sampled synthetic- (Tang et al., 2023) or real-world APIs (Xu et al., 2023; Song et al., 2023; Qin et al., 2024) and query examples. Because the relations between the sampled APIs are sparse, the resulting programs are linear sequences of often unrelated API calls. As such, tool-use corpora mostly evaluate LLMs' ability to parse API call sequences rather than complex reasoning with multiple tools. By grounding queries in a library with primitives sharing type relations, we generate more challenging and natural tasks (see §G.2) that require multi-step, arithmetic and logical reasoning, building on Shen et al. (2023), who ground queries in handcrafted task relation graphs.

LLM Agents Synthetic data generation at scale comes with quality (Iskander et al., 2024) and evaluation (Guo et al., 2024) challenges. To tackle the former, human-authoring and manual curation have been increasingly employed (Huang et al., 2024; Jhamtani et al., 2024; Yan et al., 2024, *inter alia*). Instead, we propose an interactive data generation engine to ensure data quality and reduce human cost. Like WorkBebnch (Styles et al., 2024) and AppWorld (Trivedi et al., 2024) we tackle evaluation challenges by executing agent actions in a simulated environment and determining whether they satisfy the user goal. While both WorkBench and AppWorld template user queries and resort to program templates (WorkBench) or high-fidelity task simulators (AppWorld) to annotate environment state, ASPERA does not constrain the format of the query or of the program grounding it. Like AppWorld we generalise the strict database comparisons of WorkBench, but generate the evaluation programs in LLM interactions as opposed to manually implementing them for every task.

Web agent benchmarks and *Asper-Bench* differ in action space complexity. The former typically define small action sets; for example, WebArena (Zhou et al., 2024) has 12 actions with simple descriptions like `new_tab` (*Open a new tab*). In contrast, *Asper-Bench* features 69 primitives, spanning high- (e.g., `delete_event`) and low-level ones (e.g., `now_`). This richer action space demands assistants reason over fine-grained dependencies and documentation to generate complex programs satisfying user requests. Meanwhile, web agents generate short action sequences, one step at a time, to achieve simpler goals[12]. See §G.2 for an in-depth comparison.

Code generation LLM ability is measured by benchmarks (Chen et al., 2021; Austin et al., 2021b; Hendrycks et al., 2021) which test algorithmic ability via generation of self-contained functions with contextual dependencies limited to standard libraries. To address this, other resources encompass narrow-domain dependencies on external data-science libraries (Lai et al., 2023; Wang et al., 2023) or a broader set of domains (Zhuo et al., 2024). ASPERA focuses on program generation with project-runnable dependencies (Yu et al., 2024) of custom primitives in the assistant library, which is very challenging but receives limited coverage in existing resources (Siddiq et al., 2024). Moreover, ASPERA tasks represent high-level user goals requiring the assistant to reason about primitive relevance, while the aforementioned benchmarks test program generation given precise function specifications and knowledge about external libraries acquired during pre-training. Evaluation robustness is guaranteed by execution of human-authored tests for all the above benchmarks except Zhuo et al. (2024) who, like our work, use human-LLM interaction to generate data and robustly evalute general software task competence.

8 Conclusion and Future Work

This work evaluated the ability of LLMs to parse complex natural language queries into executable programs that involve non-trivial primitive composition and flow control. We have addressed key limitations in existing work regarding dataset availability and evaluation by devising an environment where LLMs and human developers interact to collect evaluation data and code for environment state initialisation and execution outcome verification. We found that generating programs which satisfy intricate user queries grounded in custom assistant libraries is challenging for a wide range of SOTA LLMs which are otherwise proficient at code generation. Our initial results also showed that, while SOTA LLMs can compose primitives to execute complex tasks, they struggle to determine when a specific primitive is required given the query alone which is of concern to practical digital assistants. Hence, *Asper-Bench* and the ASPERA framework enable future study of action execution in the challenging setting where the primitives are not known to the agent and must be retrieved or discovered via environment interaction. Recently, agent-based approaches have been proposed to address such challenges for software engineering (Yang et al., 2024; Wang et al., 2025). Extending these approaches for complex query parsing is a promising direction for future work.

[12]Compare https://bit.ly/3CUqK1k with Figure 1 and the *Asper-Bench* AEPs in App. C.1 and our data release.

9 Limitations

Dataset size *Asper-Bench* is comparable in size to other popular code generation benchmarks such as HumanEval (Chen et al., 2021), NumpyEval (Zan et al., 2022b) , PandasEval (Zan et al., 2022b) and TorchDataEval (Zan et al., 2022a), but likely not sufficiently large for fine-tuning LLMs for digital assistant applications. Future work could focus on scaling the size of our data using the ASPERA data generation engine or by LLM-assisted paraphrasing of existing queries and refactoring of SIPs and EPs, similar to Zhuo et al. (2024). This would enable future work to study robustness of finetuned digital assistant models under non-trivial, semantics-preserving transformations of the assistant library (e.g., refactoring).

Limited domain coverage The ASPERA assistant library supports parsing of complex time expressions and a simple simulation of a corporate calendar. Furthermore, the assistant library provides documentation for 6 domains (see §4, footnote 12). With more time investment, these domains could be simulated, along with any additional simulation and evaluation tools necessary to generate the environment state. The expansion could focus on evaluating requests which span multiple applications (e.g., *How long will it take me to drive to my next meeting this afternoon?*) which are not supported in the current release.

We note that, while the simulation and the current set of evaluation and simulation tools were developed offline by one of the authors with GPT-4o assistance, future releases could explore using LLMs for assisting the developer with auxiliary tool implementation during the ASPERA interactive session. We anticipate that the human effort required to scale to new domains depends on the LLMs available for data generation, the complexity of the domain considered and the complexity of the scenarios developers wish to simulate.

Dataset bias As discussed in Sections 2.3.1 and 2.4, we mitigate dataset bias during *Asper-Bench* generation through three strategies: incorporating query history into AEP prompts, conditioning generation on focus instructions and filtering repeated or redundant examples. However, these safeguards have limitations. Large language models may not consistently follow instructions, and filtering becomes increasingly challenging as dataset size grows. As a result, the degree of bias in ASPERA-generated datasets ultimately depends on both the underlying LLM and the extent of human oversight during data curation.

Multi-turn interactions In keeping with works focused on multiple tool use and LLM agents, our work considers a user which issues a complex request in a single-turn interaction. In practice, it is desirable that the digital assistant can handle complex requests at any point in a conversation. Moreover, multi-turn interaction is necessary when the assistant cannot perform entity disambiguation or has failed to solve the task. Future work could exploit the error handling sequences in the reference *Asper-Bench* AEPs to generate dialogues where complex action execution requires user interaction, similar to recent work by Lu et al. (2024).

Human supervision requirement As discussed in §3, ASPERA currently relies on human supervision and (optional) interactive prompting to ensure the generation of high-quality and diverse data. Even state-of-the-art models such as GPT-4o (release `gpt-4o-2024-05-13`) did not reliably produce data suitable for robust capability evaluation without supervision. However, while future improvements in LLM capabilities are expected to reduce this human oversight requirement, the current framework already supports capturing developer interventions and on-the-fly annotation of natural language error descriptions. Over time, this interaction naturally creates a growing corpus of domain-specific errors and corresponding corrections, which can then be leveraged to further fine-tune and improve the performance of the data-generation models themselves.

Interactive code generation Humans write code in an interactive manner (Yang et al., 2023), occasionally relying on execution feedback to correct errors, resolve ambiguities and decompose tasks iteratively. The majority of existing code generation benchmarks, including the current work, consider a non-interactive instruction-to-code sequence transduction process which has the potential for error propagation and a disconnect between the generated code and its execution environment. While the ASPERA environment supports interactive code generation grounded in environment feedback and observations, we have focused on evaluating LLMs' fine-grained understanding and ability to compositionally use multiple primitives and curated the tasks such that that they are solvable without interaction. In doing so, we have increased the difficulty of certain types of tasks (e.g., scheduling subject to constraints, tasks involving re-scheduling and diary re-organisation). Important baselines to be considered in future work are incremental program generation following ReACT (Yao et al., 2023) framework as well as more agents specialised for software engineering discussed in §8.

Primitive selection While our primitive selection (PS) baseline partially emulates how human developers interact with unfamiliar codebases, it remains relatively simple. As discussed in §4, the agent is not given prior knowledge of available primitives at deployment, making this setting a more realistic and robust measure of an LLM's ability to execute complex tasks. Future work should explore more sophisticated strategies for hierarchical codebase exploration and incremental action execution program generation.

Efficiency Meeting real-world latency and cost constraints requires agents capable of executing complex queries within tight input and output budgets. In ASPERA, agents are prompted with `python`-style function signatures, type annotations and docstrings. Future work should investigate more compact and expressive representations of documentation such as compressed module-level summaries containing key function signatures and type relations. Additionally, for large assistant libraries, concurrent exploration by multiple agents could improve efficiency. These directions hold promise for scaling LLMs to broad virtual assistant deployment scenarios.

Scenario-based evaluation We have designed ASPERA such that each task can have multiple SIPs and corresponding EPs to support creating contrast sets (Gardner et al., 2020) for each task and comprehensively evaluate that agent actions satisfy the user goal regardless of the initial state. However, unlike in domains such as customer resource management (Styles et al., 2024) or online ordering (Trivedi et al., 2024) where the user may not know the state of the environment, we assume that the user has complete knowledge of the state of their calendar. Consequently, scenario-based evaluation is very limited in *Asper-Bench* and concerns only queries involving the calendars of other actors in the environment (e.g., other employees) or the room booking system. Moreover, we do not generate states where entities are ambiguous (e.g., two employees share the same surname and the user attempts to schedule a meeting with one of them without further identifying them). Future work could thus extend the SIP generation to support scenario-based evaluation.

Acknowledgments

Alexandru Coca's doctoral studies are supported by the EPSRC grant EP/R513180/1. He would like to acknowledge the significant contribution of Prof Bill Byrne from the University of Cambridge to the presentation of this work. The authors also thank Jian Zhang, Rohit Gupta, Ran Zmigrod, and Stephen Pulman from Apple for feedback and improvement suggestions on earlier versions of this work. We also thank the anonymous reviewers and the area chairs for their helpful and constructive improvement suggestions on our original draft.

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Andrew Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Trans. Assoc. Comput. Linguistics*, 8:556–571.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021a. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021b. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Ben Bogin, Shivanshu Gupta, Peter Clark, and Ashish Sabharwal. 2024. Leveraging code to improve in-context learning for semantic parsing. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 4971–5012. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Jianpeng Cheng, Devang Agrawal, Héctor Martínez Alonso, Shruti Bhargava, Joris Driesen, Federico Flego, Dain Kaplan, Dimitri Kartsaklis, Lin Li, Dhivya Piraviperumal, Jason D. Williams, Hong Yu, Diarmuid Ó Séaghdha, and Anders Johannsen. 2020. Conversational semantic parsing for dialog state tracking. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8107–8117. Association for Computational Linguistics.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Workflowllm: Enhancing workflow orchestration capability of large language models. *CoRR*, abs/2411.05451.

Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. 2020. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1307–1323. Association for Computational Linguistics.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 11143–11156. Association for Computational Linguistics.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

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 *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *CoRR*, abs/2308.00675.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Shadi Iskander, Sofia Tolmach, Ori Shapira, Nachshon Cohen, and Zohar Karnin. 2024. Quality matters: Evaluating synthetic data for tool-using llms. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 4958–4976. Association for Computational Linguistics.

Harsh Jhamtani, Hao Fang, Patrick Xia, Eran Levy, Jacob Andreas, and Ben Van Durme. 2024. Natural language decomposition and interpretation of complex utterances.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida I. Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 18319–18345. PMLR.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

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. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. *CoRR*, abs/2408.04682.

Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8494–8502. Computer Vision Foundation / IEEE Computer Society.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *CoRR*, abs/2405.17935.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, Jean-Baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew M. Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, James Molloy, Jilin Chen, Michael Isard, Paul Barham, Tom Hennigan, Ross McIlroy, Melvin Johnson, Johan Schalkwyk, Eli Collins, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Clemens Meyer, Gregory Thornton, Zhen Yang, Henryk Michalewski, Zaheer Abbas, Nathan Schucher, Ankesh Anand, Richard Ives, James Keeling, Karel Lenc, Salem Haykal, Siamak Shakeri, Pranav Shyam, Aakanksha Chowdhery, Roman Ring, Stephen Spencer, Eren Sezener, and et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR*, abs/2403.05530.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2023. Taskbench: Benchmarking large language models for task automation. *CoRR*, abs/2311.18760.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10737–10746. Computer Vision Foundation / IEEE.

Mohammed Latif Siddiq, Simantika Dristi, Joy Saha, and Joanna C. S. Santos. 2024. The fault in our stars: Quality assessment of code generation benchmarks. *CoRR*, abs/2404.10155.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *CoRR*, abs/2306.06624.

Pragya Srivastava, Satvik Golechha, Amit Deshpande, and Amit Sharma. 2024. NICE: to optimize in-context examples or not? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 5494–5510. Association for Computational Linguistics.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. 2024. Workbench: a benchmark dataset for agents in a realistic workplace setting. *CoRR*, abs/2405.00823.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *CoRR*, abs/2306.05301.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. 2021. Androidenv: A reinforcement learning platform for android. *CoRR*, abs/2105.13231.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 16022–16076. Association for Computational Linguistics.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024a. Llms in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August*

11-16, 2024, pages 10583–10604. Association for Computational Linguistics.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, and et al. 2025. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. 2023. Execution-based evaluation for open-domain code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 1271–1290. Association for Computational Linguistics.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. 2024b. Officebench: Benchmarking language agents across multiple applications for office automation. *CoRR*, abs/2407.19056.

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 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *CoRR*, abs/2305.16504.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`.

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. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yuekun Yao and Alexander Koller. 2022. Structural generalization is hard for sequence-to-sequence models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5048–5062, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 37:1–37:12. ACM.

Daoguang Zan, Bei Chen, Zeqi Lin, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2022a. When language model meets private library. In *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 277–288. Association for Computational Linguistics.

Daoguang Zan, Bei Chen, Dejian Yang, Zeqi Lin, Minsu Kim, Bei Guan, Yongji Wang, Weizhu Chen, and Jian-Guang Lou. 2022b. CERT: continual pre-training on sketches for library-oriented code generation. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2369–2375. ijcai.org.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro von Werra. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *CoRR*, abs/2406.15877.

A ASPERA dataset generation prompts

A.1 Joint query and AEP generation

````
My team needs your help with generation a wide variety of complex programs that can be implemented with
our application backend. We care to generate only programs that would be generated by our large language
model when interacting with our application via a voice interface.

Here is our application code.

```python
{{ code }}
```

Here are some examples of high quality programs that we wrote to help you understand the task.

```python
{{ query_solution_examples }}
```

Guidelines:

1. Please limit yourself to generating programs involving complex combinations of the members of our
codebase. It is not helpful to assume scenarios that our application cannot implement or assume unknown
details about method implementations - focus on the interfaces and read our documentation carefully.

2. Diversity is key. Focus on user requests that can be parsed to a fairly complex program implemented
with the codebase above. Just put yourself in the shoes of the user wanting to get a lot done with our
application. Some ways to achieve diversity may be:
    - imagine scenarios using for loops
    - imagine scenarios based on user conditions
    - imagine scenarios requiring filtering operations
    - imagine many scenarios where multiple dataclasses and their methods are required to support a
      complex user goal
    - scenarios imagined should always be compositional (ie always have diverse combinations of object
      attributes and methods operating on them)

3. To reiterate, diversity (2) should not come at the expense of imagining scenarios our codebase cannot
support (1). We will discuss how to improve our codebase in the future.

### Program structure guidelines ###
The examples above follow {{ guidelines.generation_labelling | length }} structure guidelines listed
below. Do the same, clearly stating when you follow them in your comments, as demonstrated above.
{% for instruction in guidelines.generation_labelling %}
{{ loop.index }}. {{ instruction }}
{%- endfor %}
````

Figure 6: System turn. In the above the field code is replaced with the documentation of the assistant library and `query_solution_examples` is replaced with 5 AEP examples. See Figure 8 for guidelines definition.

```
You have done a stellar job generating some brilliant programs and user queries already. To remind you
of work you completed and keep things brief, we only show the queries extracted from the docstrings of
programs you generated:

{% for q in queries %}
{{ loop.index }}. {{ q }}
{%- endfor %}

Now we have to generate more programs representing complex user utterances. Crucially, these should
represent a complex set of new user queries, where the user tries to complete different tasks from
the ones you generated above. *Do not merely paraphrase the queries you already generated* when
synthesizing programs - think of new and original complex user tasks that our application backend
supports.

{% if focus -%}
{{ focus }}
{% endif -%}

Let us generate {{ n_programs }} programs.

```python"""
```

Figure 7: User turn. To encourage diversity, we optionally include the history of the queries generated in the prompt, similar to Wang et al. (2024a). If `n_programs` is set to values greater than 1, multiple programs are generated. The `focus` field can be changed after each round of interaction, to encourage diversity of generated queries and programs.

```
• Employee names are generally assumed unique, so you may use find_employee(name)[0] for resolving
  a name to an Employee object. Use this sparingly; even though there may be multiple employees with
  the same name, the user query might give additional information which resolves the ambiguity (eg
  specify the meeting time). If you decided to make this assumption add a 'by structure guideline
  #1' comment.
• Work meetings can start after 9:06 AM and should end before 5:10 PM. They don't happen at the
  weekend unless the user explicitly mentions so.
• Type annotate the return for programs which have a return type which is not None.
• Do not call functions with default optional values.
```

Figure 8: Guidelines iterated over to populate `{{instruction}}` fields in the loop in Figure 6. The first guideline enforces a unique entity name environment constraint, which grounds 0-indexing `find_employee` results. We make this design decision to decrease task difficulty for our initial release, but note the LLM is instructed to mark this assumption with `# by structure guideline 1` to support future LLM-based annotations of AEPs which handle disambiguation. The second guideline encodes a simple events scheduling policy to be followed when explicit constraints are not provided by the user and when rescheduling events. The third guideline prompts for return type annotation for information-seeking queries and the final guideline encourages concise coding.
```

A.2 AEP generation given human-authored request

```
You are an expert programmer working with my team which is specialising in developing AI assistants. Your current task is to translate a series
of complex user requests into executable `python` programs using our application backend below:

```python
{{ code }}
```

Here are some examples your colleagues shared with you to help you generate your response in a style that is compatible with our infrastructure:

```python
{{ query_solution_examples }}
```

{% if guidelines.generation_labelling -%}
Program structure guidelines
The examples above follow the {{ guidelines.generation_labelling | length }} structure guidelines listed below. Do the same, clearly stating when you
follow them in your comments, as demonstrated above.
{% for instruction in guidelines.generation_labelling %}
{{ loop.index }}. {{ instruction }}
{%- endfor %}
{%- else %}
{%- endif %}
```

(a) System turn. The code and `query_solution_examples` fields are populated with the assistant library documentation and 5 AEP examples like in the joint AEP and query generation prompt depicted in Figure 6.

```
Now it's your turn. Please translate the queries below into `python` programs using the examples above to guide your response format. The response should be
inside a Python markdown block.
{% for q in queries %}
{{ loop.index }}. {{ q }}
{%- endfor %}

```python"""
```

(b) User turn. The framework supports AEP generation for query batches.

Figure 9: Prompt template used for AEP generation given a human-authored request. (Section 2.3.1)
```

A.3 SIP generation

```
For testing purposes, we need to generate the underlying runtime  state of the user device. Your task is to carefully analyse
`{{ plan_name }}` along with the application code above and assist our testing team in setting up the runtime environment such
that `{{ plan_name }}` can be executed and its outputs verified. To do so, you will need to generate a `python` function named `{{ setup_function_name }}`.

We have implemented additional tooling you may find helpful for completing this task:

```python
{{ setup_code }}
```

You may use additional knowledge and create your own functions if needed - custom functions should be defined inside the
`{{ setup_function_name }}` function. Note how we import modules in the standard python library locally inside the
`{{ setup_function_name }}` and how our application code does not need to be imported (we automatically do so when we run the code).

Here are some comprehensive examples your testing team colleagues shared to help you generate a high quality program that sets up
the runtime environment correctly.

```python
{{ runtime_setup_examples }}
```

{% if guidelines.runtime_setup -%}
Runtime environment setup guidelines
The examples above follow the {{ guidelines.runtime_setup | length }} setup guidelines listed below. Do the same, clearly stating when
you follow them in your comments, as demonstrated above.
{% for instruction in guidelines.runtime_setup %}
{{ loop.index }}. {{ instruction }}
{%- endfor %}
{%- else %}
{%- endif %}

Let's now write `{{ setup_function_name }}`, our developers wrote some TODOs to get you started.

```python
def setup_env_{{plan_name}}():
    """Simulate the environment for the query:

    {{ query }}

    Note this means to create any persons, contacts, emails, events and everything that should exist
    in the user's virtual context when they make the query. You **should not** create new entities that are
    implied in the user request that the assistant has created in the `{{plan_name}}` function.
    """
'''
    {{ TODOs }}
```

(a) User turn for SIP generation. This turn is added to the chat history which contains the AEP generation system and user turns and assistant turn with the generated AEPs. `plan_name` is the name of the AEP function for which the state is to be initialised and the `setup_function_name` is the name of the SIP to be generated. `setup_code` is replacted by the documentation for additional tools the LLM can call to simulate complex environment state. One example is `simulate_org` in Figure 2 (program B, l. 9 - 11) which allows the LLM to simulate an organisation with a complex reporting structure by parametrising the simulation. The `runtime_setup_examples` field shows 5 SIP examples, which initialise the state for the 5 AEP examples in the chat history. Guidelines, shown in Figure 10b, state simulation assumptions. The LLM is prompted to mark these assumptions in comments to enable LLM-assisted refactoring of the SIPs. The `query` field is replaced by the user query. The `TODOs` fields marks instruction the developer may optionally specify. These are formatted on separate lines following `#TODO:` tags.

- Dates should be grounded using the tools in the time_utils library. When doing so, add a 'setup guideline #1' comment.
- Work meetings can start after 9:06 AM and should end before 5:10 PM. When doing so, add a 'setup guideline #2' comment.
- Events assumed to occur in the future should start after the date and time specified by time_utils.now_(), whereas events in the past should finish before time_utils.now_(). When doing so, add a 'setup guideline #3' comment.
- Employee names are assumed unique, so you may use find_employee(name)[0] for resolving a name to an Employee object. When doing so, add a 'setup guideline #4' comment.
- Ensure you follow all the TODOs with appropriate steps, but don't be afraid to do additional steps if you think it necessary - our developers may not write detailed enough TODOs.

(b) Guidelines used to populate `{{instruction}}` in the bottom loop of (a).

Figure 10: Prompt template used for runtime setup program generation (Figure 2, B).
```

A.4 EP generation

```
We need some test code to check that `{{ plan_name }}` executes correctly on the user device. After a careful analysis of `{{ plan_name }}` and
`{{ setup_function_name }}` (defined below), your task is to write a function `{{ test_function_name }}` to do so.

We have implemented additional tooling you may find helpful for completing this task:

```python
{{ setup_code }}
```

```python
{{ testing_code }}
```

You may use additional knowledge and create your own functions if needed - custom functions should be defined inside the `{{ test_function_name }}`
function. Note how we import modules in the standard python library locally inside the s`{{ test_function_name }}` and how our application code does not
need to be imported (we automatically do so when we run the code).

Here are some comprehensive examples your testing team colleagues wrote:

```python
{{ evaluation_examples }}
```
{% if guidelines.evaluation -%}
Testing guidelines
The examples above follow the {{ guidelines.evaluation | length }} setup guidelines listed below. Do the same, clearly stating when you
follow them in your comments, as demonstrated above.
{% for instruction in guidelines.evaluation %}
{{ loop.index }}. {{ instruction }}
{%- endfor %}
{%- else %}
{%- endif %}

 Here is the code that sets up the runtime environment for `{{ plan_name }}` execution:

 ```python
 {{ runtime_setup_program }}
 ```

Write `{{ test_function_name }}`:

```python
def evaluate_{plan_name}(
    query: str, executable: Callable[[], Any], setup_function: Callable[[], Any]
):
    """Validate that `executable` program for the query

    {{ query }}

    has the expected effect on the runtime environment.

    Parameters
    ----------
    query
        The query to validate
        executable
        The query execution function, `{plan_name}`
    setup_function
        `{setup_function_name}` function.
    """
'''
```

(a) User turn template for EP generation. This turn is added to the chat history, which contains at this point the user and system turns for AEP and SIP generation. `plan_name`, `setup_function_name` and `test_function_name` are formatted with the AEP, SIP and EP function names, respectively. `setup_code` is defined in Figure 10 and `testing_code` is replaced by documentation of other tools the LLM can use to verify AEP correctness (see App. A.5). The `evaluation_examples` field is replaced by 5 EP examples, which demonstrated how to evaluate the correctness of the AEPs in the interaction history given the SIPs examples. Guidelines, shown in Figure 11b, provide relevant assumptions for writing correct and concise test code (App. A.5). The LLM is prompted to mark these assumptions in comments to enable LLM-assisted refactoring of the EPs. The `runtime_setup_program` is the SIP, and `test_function_name` is name of the EP to be generated.

- fields of type list[Employee] of events returned by find_events are sorted alphabetically according to the name attribute. Sort attendees lists you create accordingly. When doing so, add a 'testing guideline #1' comment"
- For queries that have a return type, consider a range of possible alternative return types that could have been returned instead by the executable and check the result correctness in those cases too. Add a '#testing guideline #2' comment in this case.
- When checking events requested by the user were created, never test equality of the 'subject' attribute because variations in the meeting name can affect test robustness.
- When add_event is called without an ends_at parameter, a default duration of 16 minutes is assumed when writing the event to the underlying database. Check that the events for which end time or duration is not specified satisfy this constraint.
- SolutionError message is always 'Incorrect Solution'.
- Where possible, use the information in the runtime environment setup function below to simplify testing code.

(b) Guidelines.

Figure 11: Prompt template used for evaluation program generation (Figure 2, C).
```

A.5 Auxiliary ASPERA tools

ASPERA defines auxiliary tools designed to aid SIP and EP generation (Table 9). These can be implemented by the developer interactively[13] or (before task generation begins).

Module	Tool	Functionality
Simulation Tools		
work_calendar	simulate_user_calendar	Adds a set of LLM-generated events to the user's calendar.
	simulate_employee_calendar	Adds a set of LLM-generated events to the calendar of a given employee.
company_directory	simulate_org_structure	Build an organisation structure given, employee names, team membership, user name and user role. Reporting relationships and employee profiles are simulated by ASPERA.
	simulate_vacation_schedule	Simulate the vacation schedule of a given employee.
	UserRole	Enum listing key company roles such as CEO and COO.
room_booking	simulate_conference_room	Add a conference room to the conference room database.
Evaluation Tools		
Module	**Tool**	**Functionality**
time_utils	repetition_schedule	Create a recurrence schedule for a meeting or reminder.
work_calendar	assert_user_calendar_shared	Check that a calendar has been shared between a list of employees.

Table 9: ASPERA auxiliary tools.

Simulation tools Simulation tools are included in SIP generation prompts to allow the LLM to create entities stored in environment databases. These tools differ in their implementation complexity. Some tools (e.g., `simulate_user_calendar`) simply write LLM-defined entities to the environment databases whereas others can be used to invoke more advanced simulations implemented by developers (possibly with LLM assistance) in ASPERA (e.g., `simulate_org_structure`). The LLM uses information in the query and the AEP to parametrise the simulation and generates complex entities as a result.

```python
class RepetitionSpec(BaseModel):
    frequency: EventFrequency
    period: int = 1
    recurs_until: datetime.date | datetime.datetime | None = None
    max_repetitions: int | None = None
    which_weekday: list[int] | None = None
    which_month_day: list[int] | None = None
    which_year_month: list[int] | None = None
    bysetpos: list[int] | None = None
    exclude_occurrence: list[datetime.datetime] | None = None
    occurrence_on_date: datetime.datetime | None = None
```

Figure 12: Definition of `RepetitionSpec`, an object used for generating recurring event instances. Documentation omitted for brevity.

Evaluation tools EP generation prompts include evaluation tools to support robust evaluation and access to environment state that is not possible with the tools the assistant uses to compose AEPs. To understand why this is necessary, consider the query *Remind me to check arxiv on Wednesdays.* To execute this action, the assistant must create an `Event` instance and set the `repeats` property to a correctly parametrised recurrence rule (a `RepetitionSpec` instance, shown in Figure 12). Because the recurrence always inherits the parent event parameters, setting `which_weekday=[2]` in this case is optional. More generally, complex recurrences admit multiple parametrisations which are difficult to enumerate for developers. For this reason, we include the `repetition_schedule` tool in the prompt so that the LLM can use it to compare the event instances it returns rather than comparing generator object properties. This ensures robust comparison independent of `RepetitionSpec` parametrisation.

[13]The developer is prompted to implement simulation tools after AEP generation and evaluation tools after SIP generation. The implemented tools are displayed in the subsequent SIP/EP generation prompts.

B Assistant library

	time_utils	work_calendar	company_directory	room_booking
Functions				
	now_	get_default_preparation_time	get_current_user	find_available_time_slots
	get_weekday	add_event	find_employee	room_booking_default_time_window
	this_week_datestextsuperscript*	find_events	find_team_of	search_conference_room
	get_weekday_ordinaltextsuperscript*	find_past_events	find_reports_of	summarise_availabilitytextsuperscript*
	parse_time_stringtextsuperscript*	get_calendar	find_manager_of	
	time_by_hmtextsuperscript*	delete_event	get_assistant	
	date_by_mdytextsuperscript*	get_search_settings	get_vacation_schedule	
	get_next_dowtextsuperscript*	find_available_slotstextsuperscript	get_employee_profile	
	get_prev_dowtextsuperscript*	share_calendar	get_all_employees	
	parse_duration_to_calendartextsuperscript*	summarise_calendar	get_office_location	
	parse_durations_to_date_intervaltextsuperscript*	provide_event_details		
	parse_date_stringtextsuperscript*			
	sum_time_unitstextsuperscript*			
	compare_with_fixed_duration			
	modifytextsuperscript*			
	combine			
	intervals_overlaptextsuperscript*			
	replacetextsuperscript*			
Objects				
	Duration	Eventtextsuperscript	EmployeeDetails	ConferenceRoom
	TimeInterval	CalendarSearchSettings	Employee	RoomAvailability
	DateRange			
	RepetitionSpec			
Enums				
	TimeExpressions	ShowAsStatus	Team	
	DateRanges			
	DateExpressions			
	TimeUnits			
	DateTimeClauseOperators			
	ComparisonResult			
	EventFrequency			

Table 10: The ASPERA assistant library defines 62 primitives across 4 domains, implemented by a single developer with GPT-4o assistance. Primitives marked with * were implemented interactively with the LLM using the ChatGPT graphical user interface. For each primitive, the LLM was prompted with the docstring describing the primitive functionality, and its output subsequently refined until the specification was correctly implemented, if necessary. Unit tests were generated in addition to developer-authored tests to verify complex functionality. Primitives marked with †were implemented with partial LLM assistance, where the developer described the functionality to be implemented to the LLM, but substantially refactored and enhanced the code. The LLM was also used for generating unit tests for †primitives.

C Dataset characterisation

C.1 Examples of challenging tasks

```python
def check_boss_availability() -> bool:
    """Check the boss' calendar from Wednesday to Friday next week for availability."""

    # find the current user's manager
    current_user = get_current_user()
    manager = find_manager_of(current_user)

    # calculate the dates for Wednesday to Friday next week
    next_week_monday = get_next_dow("Monday", after=now_().date())
    next_week_wednesday = get_next_dow("Wednesday", after=next_week_monday)
    next_week_thursday = get_next_dow("Thursday", after=next_week_monday)
    next_week_friday = get_next_dow("Friday", after=next_week_wednesday)

    # get the manager's calendar events between Wednesday to Friday next week
    all_events = get_calendar(manager)
    relevant_events = [
        event for event in all_events
        if next_week_wednesday <= event.starts_at.date() <= next_week_friday
    ]
    # check if there are some events that may cover the entire interval
    for e in all_events:
        if (
            (e.starts_at.date() < next_week_wednesday <= e.ends_at.date())
            or (next_week_wednesday <= e.starts_at.date() <= next_week_friday)
            and (next_week_friday < e.ends_at.date())
        ):
            relevant_events.append(e)
    available_slots = []
    for date in [next_week_wednesday, next_week_thursday, next_week_friday]:
        available_slots += find_available_slots(relevant_events, date=date)
    return bool(available_slots)
```

(a) *Assistant, check my boss' calendar Wednesday to Friday next week, are they available for a meeting?* Solving this query involves reasoning about time and having the common sense to account for events spanning multiple days.

```python
def schedule_strategy_review():
    """Schedule a strategy review with the CFO and the COO."""

    # find the CFO and COO
    all_employees = get_all_employees()
    leadership = [
        e
        for e in all_employees
        if get_employee_profile(e).team == Team.Leadership
    ]
    # ceo does not report to anyone
    cfo_coo = [e for e in leadership if find_manager_of(e)]
    # determine the event start time and duration
    one_week_from_today = get_next_dow("Tuesday")
    meeting_time = time_by_hm(hour=2, minute=30, am_or_pm="pm")
    starts_at = combine(one_week_from_today, meeting_time)
    duration = Duration(number=1, unit=TimeUnits.Hours)
    ends_at = modify(starts_at, duration, operator=DateTimeClauseOperators.add)
    # add the event to the calendar
    event = Event(
        subject="Strategy review with CFO and COO",
        starts_at=starts_at,
        ends_at=ends_at,
        attendees=cfo_coo,
    )
    add_event(event)
```

(b) *Assistant, add a strategy review with the CFO and the COO one week from today at 2:30 PM, for 1 hr.* Solving this query involves clever tool use to find the leadership team while taking care to exclude the CEO.

```python
def who_is_busiest_next_week() -> str:
    """Determine which of Bill or Bob is busiest next week."""

    from collections import defaultdict

    def calculate_duration(
        duration_map: dict[datetime.date, list[Duration]]
    ) -> Duration:

        def to_minutes(d: Duration) -> float:
            """Convert the Duration to minutes."""
            if d.unit == TimeUnits.Hours:
                return float(d.number * 60)
            elif d.unit == TimeUnits.Minutes:
                return float(d.number)
            elif d.unit == TimeUnits.Days:
                return float(d.number * 24 * 60)
            elif d.unit == TimeUnits.Months:
                raise TypeError("Cannot convert variable durations to minutes!")
            else:
                raise ValueError(f"Unsupported time unit: {d.unit}")
        total_minutes = 0
        for day, durations in duration_map.items():
            # the largest unit of time is returned for the sum, need
            # to make sure the units are consistent
            this_day_total = to_minutes(sum_time_units(durations))
            total_minutes += this_day_total
        return Duration(total_minutes, unit=TimeUnits.Minutes)

    # Find the employees named Bill and Bob
    bill = find_employee("Bill")[0]  # by structure guideline #1
    bob = find_employee("Bob")[0]  # by structure guideline #1

    # Get their events for next week
    next_week = parse_durations_to_date_interval(DateRanges["NextWeek"])
    bill_events = get_calendar(bill)
    bob_events = get_calendar(bob)

    # Create look-ups for relevant events in the next week
    bill_events_by_day = defaultdict(list)
    for e in bill_events:
        if next_week.start <= e.starts_at.date() <= next_week.end:
            bill_events_by_day[e.starts_at.date()].append(e.duration)

    bob_events_by_day = defaultdict(list)
    for e in bob_events:
        if next_week.start <= e.starts_at.date() <= next_week.end:
            bob_events_by_day[e.starts_at.date()].append(e.duration)

    bill_total_duration = calculate_duration(bill_events_by_day)
    bob_total_duration = calculate_duration(bob_events_by_day)

    # Compare durations and return the name of the busiest person
    if bill_total_duration.number > bob_total_duration.number:
        return "Bill"
    elif bob_total_duration.number > bill_total_duration.number:
        return "Bob"
    else:
        return "Both are equally busy"
```

(c) *Assistant, I need to know which of Bill or Bob is busiest next week so I can allocate work.* Here, summing the event duration involves careful unit conversion to provide the correct answer.

Figure 13: Challenging queries from lines 3 -5 of Table 2 as particularly challenging. Figures 13a, 13c and 13b show the sample solutions for these queries respectively, with explanations of their difficulty.

C.2 Further corpus descriptive statistics

Here, we present some further descriptive statistics of *Asper-Bench*. Tables 11 and 12 show some example queries organised according to their complexity, whereas Figures 14 to 17 show how key program complexity measures vary with query length and the distribution of *Asper-Bench* reference AEPs.

Query	**Cyclomatic complexity**	**σ from mean**
Assistant, can you tell me when are my manager and skip manager both available on Friday?	1.00	-1.14
Assistant, schedule an urgent meeting with my manager now.	1.00	-1.14
Assistant, schedule a project meeting with my team next Wednesday at 2 PM and block 30 minutes right before for preparation.	1.00	-1.14
Assistant, schedule a project update meeting with my manager before 3 PM tomorrow.	5.00	-0.16
Assistant, schedule a meeting in the afternoon with my engineering colleagues, avoiding any engineering management.	6.00	+0.08
Assistant, remove my second holiday notification from the calendar, something came up.	7.00	+0.32
Assistant, send out a meeting invite to the entire team for a company update next Monday at 2 PM, but exclude those who are on vacation.	7.00	+0.32
Assistant, see if my boss' boss and Jane have accepted my meeting request for tomorrow. If anybody declined, reschedule to take place later but at the earliest available time for everyone, I'm free all day.	19.00	+3.24
Assistant, tell me which days is Sally in office in the third week of August?	20.00	+3.48
Assistant, is there a time in August where everyone from finance is off?	21.00	+3.72

Table 11: Sampling of queries according to cyclomatic complexity of sample solution.

Query	**# unique primitives**	**σ from mean**
Assistant, how many meetings with Jianpeng are in my calendar at the moment?	2	-1.79
Assistant, cancel everything but the important meetings.	2	-1.79
Assistant, find the names of our assistants please.	2	-1.79
Assistant, schedule a meeting with my manager tomorrow at 10 AM if I have no other meetings then.	9	+0.04
Assistant, provide a summary of my manager's calendar for the next two weeks.	9	+0.04
Assistant, invite the entire sales department to a meeting today from 3 to 5.	9	+0.04
Assistant, schedule a team meeting next Monday at 10 AM, and book a conference room for it. Also, schedule a follow-up meeting one week later at the same time and book the same room.	18	+2.39
Assistant, can you schedule a 30 mins recurring weekly meeting with the engineering team on Fridays at 3 PM for the next two months? If there are clashes, tell me their dates, don't double book.	19	+2.65

Table 12: Sampling of queries according to number of unique primitives in sample solution.

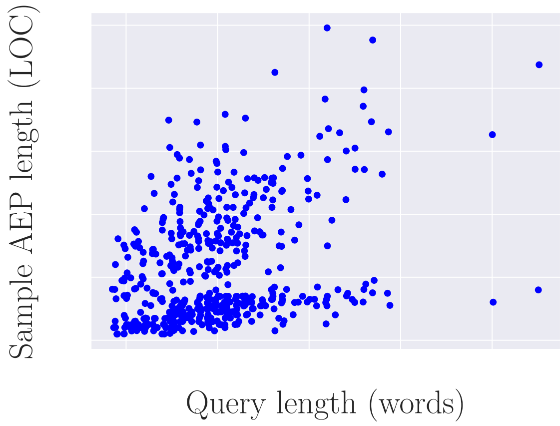


Figure 14: *Asper-Bench* AEP query length vs program length.

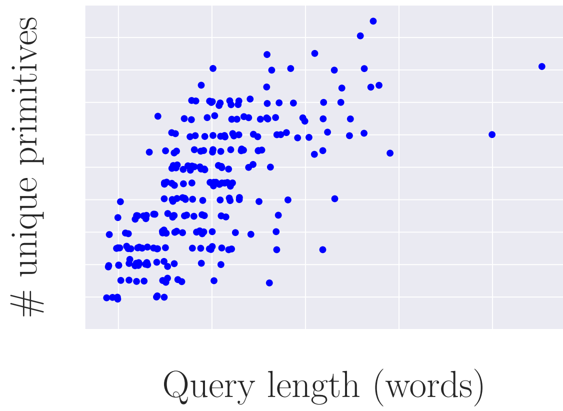


Figure 15: *Asper-Bench* AEP query length vs number of unique primitives.

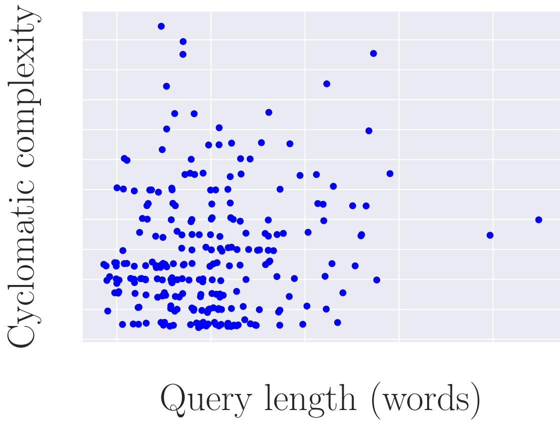


Figure 16: *Asper-Bench* AEP query length vs cyclomatic complexity.

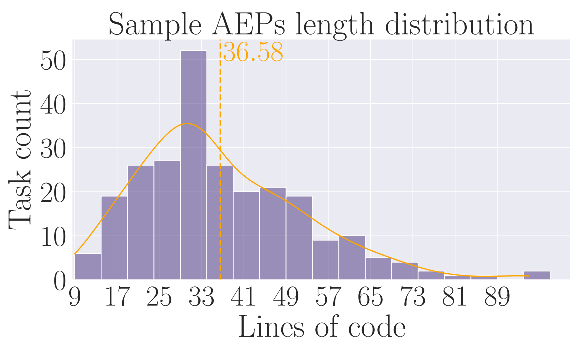


Figure 17: *Asper-Bench* AEP length distribution.

C.3 ASPERA policy

Asper-Bench programs follow a policy for interrupting execution to interact with the user: the `RequiresUserInput` exception is raised if the entities mentioned by the user cannot be retrieved from the databases[14] or the task cannot be completed (e.g., a room is unavailable), as shown in Figure 18a.

```python
def reschedule_monthly_sync():
    """Reschedule the monthly sync meeting."""

    # find the monthly sync meeting
    events = find_events(subject="Monthly Sync")
    # by structure guideline #2
    if len(events) == 0:
        raise RequiresUserInput("No monthly sync meetings found.")
    elif len(events) > 1:
        raise RequiresUserInput(
            f"{len(events)} monthly sync meetings found. Please provide more details."
        )
    else:
        monthly_sync = events[0]

        # from now_() we now it is Tuesday 25th, and 1st Monday of next month is actually next monday
        first_monday_next_month = get_next_dow("Monday")
        new_meeting_time = time_by_hm(hour=9, minute=0, am_or_pm="am")
        new_meeting_start = combine(first_monday_next_month, new_meeting_time)
        new_meeting_end = modify(
            new_meeting_start,
            monthly_sync.duration,
            operator=DateTimeClauseOperators.add
        )

        # update the event
        monthly_sync.starts_at = new_meeting_start
        monthly_sync.ends_at = new_meeting_end

        # add the updated event to the calendar
        add_event(monthly_sync)
```

(a) LLM-generated policy for error handling and disambiguation.

```python
class RequiresUserInput(Exception):
    """An exception to be raised when an assumption has to be made in order
    to continue the program. Typical situations involve:

        - indexing search results when multiple or no search results are returned
        but the user query implies the uniqueness of the search results.
        - inform the user a task could not be completed (eg no suitable meeting
        room could be booked for a meeting as per the user request).


    For additional details, consult program structure guidelines below (displayed
     only if any special guidance applies).

    Notes
    -----
    1. Cases when no or multiple search results are returned should have distinct messages
     to indicate why the error was raised.
    2. The number of returned results should always be stated when the error is raised because
     a search returned more than one result.
    3. This error *should not* be raised to respond to user questions - an appropriate object
    should be returned instead. However, if the question cannot be answered (eg user asks
    about meeting which cannot be found), then this exception can be raised.
    4. If the user warns the assistant of possible conflicts when making their request (eg
    schedule something if it doesn't conflict with something else), this error should not be raised.
    5. Employee names can be assumed unique unless the user request implies this is not the case. In this
    case it is not necessary to raise this error and raising it will have no effect."""
```

(b) `RequiresUserInput` documentation

Figure 18: ASPERA employs exceptions to generate reference AEPs following a simple policy: the assistant raises `RequiresUserInput` if a task cannot be completed due to environment constraints or if the user must disambiguate. We observe 144 `RequiresUserInput` usages across 78 programs. Additionally, top guidelines in Figure 8 enforce a simple scheduling policy.

[14]Given the complexity of our tasks, we always simulate these entities; we leave adversarial user behaviour (e.g., the user deliberately requests to update an event that is not in the calendar) to future work.

D ASPERA evaluator prompt templates

```
You are an expert programmer working with my team which is specialising in developing AI assistants. Your current task is to translate a complex user request into a `python` program using our application backend below:

```python
{{ code }}
```

Here are some examples your colleagues shared with you to help you to understand the solution format and some assumptions about our application backend.

```python
{{ query_solution_examples }}
```

The examples above follow the {{ guidelines.generation_labelling | length }} structure guidelines listed below. You must adhere to these when writing your solution.
{% for instruction in guidelines.generation_labelling %}
{{ loop.index }}. {{ instruction }}
{%- endfor %}
```

(a) Prompt template for AEP generation, shared by CCK and PS agents. The syntax (`{{ variable }}`, `{% loop %}`) follows standard templating convention, where placeholders represent dynamically inserted content and loops iterate over a list of instructions. See guidelines below.

```
• Unless the user explicitly states, meetings should not be scheduled on or recur during weekends.
• Work meetings can only happen during the times prescribed in the time_utils library unless the user explicitly states otherwise.
• The leadership team is formed of a CEO, COO, CFO. Department heads report to either the COO or the CFO.
• Use the tools in the time_utils library to reason about time. Hence, current date and time on the user device should be found using the tools and documentation in this library and not the datetime library.
• Information-seeking queries should return an appropriate object to the caller; avoid simply printing the information inside your solution.
• If you need to format dates in a string, use strftime('%Y-%m-%d'). For datetime objects use strftime('%Y-%m-%d %H:%M:%S').
• Make sure to escape \n characters.
• Type annotate the return for programs which have a return type which is not None
• Only the first Python markdown block will be executed, so if you wish to use helper functions, these should be defined locally inside your solution.
• Only import modules from the standard library that you need for your programs (eg import collections). Imports from our application backend will be automatically done when we execute the program you generate.
```

(b) The first two guidelines implement a simple events schedule policy. The third provides additional information about the environment, required to solve a range of queries involving the organisation leadership. The remainder of the guidelines are concerned with various aspects of the AEP structure such as time grounding, return type, function nesting and importing. These guidelines were designed to minimise execution errors due to mismatches between the simulation environment and model behaviour following detailed error analyses on initial agent development iterations.

Figure 19: ASPERA AEP generation prompt template.

D.1 Primitive selection prompt

```
You are a programmer using a Python library of personal assistant tools in order to write a program that executes a user query. You will be shown signatures from a Python module and a query, and will be asked to formulate Python import statements importing any tools that might be relevant to writing a program that executes the user query.

When writing the program, you will be asked to follow the {{ guidelines | length }} structure guidelines listed below.
{% for instruction in guidelines %}
{{ loop.index }}. {{ instruction }}
{%- endfor %}
Use this additional information to guide your import decisions.

Module:
{{ module }}
Query: {{ query }}

Think carefully, and output the relevant Python import statements, or None. Any code you write must be included in a Python markdown block (ie start with a "```python" sequence and end with "```"). If there are no relevant tools in the current module being shown, simply output None.
```

(a) Primitives selection prompt template.

```
• Use the tools in the time_utils library to reason about time. Hence, current date and time on the user device should be found using the tools and documentation in this library and not the datetime library.
• Work meetings can only happen during the times prescribed in the time_utils library unless the user explicitly states otherwise.
• The leadership team is formed of a CEO, COO, CFO. Department heads report to either the COO or the CFO. Appropriate tools will have to be imported and combined to resolve these employees to Employee objects required by all APIs.
```

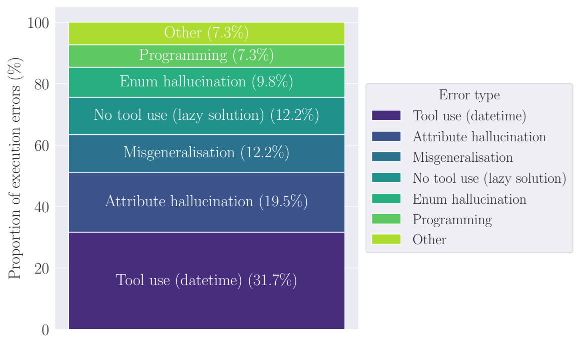
(b) Guidelines presented to the agent during at each primitive selection iteration step. These are a subset of the guidelines defined for the CCK prompt in Figure 19b, including only the instructions which can influence primitive selection.

Figure 20: Primitive selection prompt

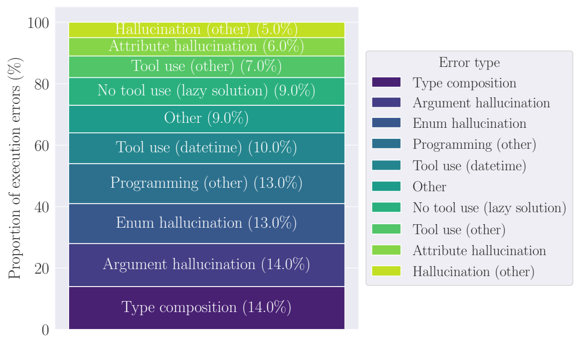
E Analysis supplementary material

E.1 Execution errors

We debug the AEPs generated by the best GPT-4o and GPT-3.5-turbo runs[15] for the first 125 queries in our corpus (50% of the data), analysing a total of 141 execution errors (Table 6) which we classify into several categories depicted in Figure 21 and for which representative examples are shown and explained in Table 13. We find execution errors occur because the LLMs hallucinate in preference to performing additional problem solving steps. While GPT-4o fails to appropriately combine the primitive to perform non-trivial compositions for date and time reasoning (row 1, Table 13) or simple arithmetic reasoning (row 5), GPT-3.5-turbo additionally fails to appropriately exploit type relations to compose primitives (row 6) and often hallucinates API arguments (row 7), demonstrating very limited ability to program according to a complex set of constraints defined by an assistant library.



(a) GPT-4o



(b) GPT-3.5-turbo

Figure 21: Execution error classification for the first 125 *Asper-Bench* queries.

[15] Success rate of 46% and 11.2%, respectively.

Id	Query	Error Snippet

1 — Assistant, schedule our team Christmas party 10 days before Christmas. Should start in the morning and end at 10 PM.

```
1  def schedule_team_christmas_party():
2
3      # find the user's team to determine event attendees
4      ...
5
6      # resolve the date for 10 days before Christmas
7      christmas_day = parse_date_string(DateExpressions.ChristmasDay)
8      party_date = modify(
9          christmas_day,
10         Duration(10, TimeUnits.Days),
11         operator=DateTimeClauseOperators.subtract,
12     ).date()
```

Tool use (datetime): Line 9 contains a `TypeError`, `modify` only accepts `datetime` objects. A correct solution requires an additional reasoning step: pass `christmas_day` and one of specified times to the `combine` library function to get the correct type.

2 — Assistant, set up a training session for all employees from the Engineering team next Monday from 2 PM to 5 PM. Send out invites and book a conference room that fits 20 people.

```
1  def schedule_engineering_training_session():
2
3      # Find all employees in the Engineering team
4      engineering_team = [
5          emp for emp in get_all_employees()
6          if emp.team == Team.Engineering
7      ]
```

Attribute hallucination: In, line 6 the `.team` attribute access raises an error because the `Employee` objects returned by `get_all_employees` only have `name` as attribute. The `Employee` object should be passed instead to the `get_employee_profile` library function to return an object which has `team` as an attribute.

3 — Assistant, can you schedule a 30 mins recurring weekly meeting with the engineering team on Fridays at 3 PM for the next two months? If there are clashes, tell me their dates, don't double book.

```
1  from typing import list
2
3  def schedule_weekly_meeting_with_engineering_team() -> (
4      list[datetime.date] | None
5  ):
6
```

Misgeneralisation: The assistant triggers an import error in line 1. The pretraining data contains `from typing import List`, a common idiom for static typing prior to PEP 585 (2019). When prompted to return an object of type `list[datetime.date] | None`, the model does not make this distinction and misgeneralises by generating line 1.

4 — Assistant, put 45 minutes in the calendar, back-to-back, with Engineering and Marketing starting at 10 AM tomorrow... Actually, add a 10-minute buffer between each meeting.

```
1  def schedule_back_to_back_meetings():
2
3      # find the user's profile
4      ...
5
6      # find the teams
7      engineering_team = find_team_of(Employee(name="Engineering"))
8      marketing_team = find_team_of(Employee(name="Marketing"))
```

No tool use (lazy solution): The assistant hallucinates lines `7-8` instead of using relevant APIs to find the engineering team, in spite of documentation that states that `Employee` objects cannot be instantiated. The functions `get_all_employees`, `get_employee_profile` and the enumeration `Team.Engineering` should have been composed, similar to snippet in row 2.

5 — Assistant, mark my vacation from next Tuesday for 2 weeks and cancel all my meetings during this period.

```
1  def mark_vacation_and_cancel_meetings():
2
3      # Determine the vacation start and end dates
4      next_tuesday = get_next_dow("Tuesday")
5      vacation_start = next_tuesday
6      vacation_end = modify(
7          vacation_start,
8          Duration(2, TimeUnits.Weeks),
9          operator=DateTimeClauseOperators.add,
10     ).date()
```

Enum hallucination: The assistant uses the enum value `TimeUnits.Weeks` (line 8), which is undefined. The library deliberately defines the `TimeUnits` members as "Hours", "Minutes", "Days", "Months" so that assistants have to perform simple unit conversions.

6 — Assistant, notify me of overlapping meetings this week.

```
1  def notify_overlapping_meetings_this_week() -> list[Event] | None:
2      # Find all future events in the user's calendar
3      ...
4      # Create a list to store overlapping meetings
5      ...
6      # Get the dates for the current week
7      ...
8      # Create a dictionary to store events by date
9      events_by_date = defaultdict(list)
10
11     # Populate the events_by_date dictionary with events happening
12     ...
13
14     # Check for overlapping meetings
15     for date, events in events_by_date.items():
16         for i, event1 in enumerate(events):
17             for event2 in events[i + 1 :]:
18                 if intervals_overlap(event1, event2):
```

Type composition: The assistant calls `intervals_overlap` with `Event` instead of `TimeInterval` types (line 18). The latter must be instantiated from the event properties.

7 — Assistant, block time for preparation before important meetings.

```
1  def block_preparation_time_for_meetings():
2      # Find the user's upcoming important meetings
3      user = get_current_user()
4      important_meetings = find_events(attendees=[user], event_importance="high")
```

Argument hallucination: The assistant calls `find_event` with `event_importance` keyword (line 4). Valid `find_event` arguments are `attendees` and `subject`.

Table 13: Sample execution errors.

E.2 Task completion error examples

Id	Query	Agent action
1	Assistant, Ari and James are on holiday next month, who's out for longer?	Sums duration of all vacations, month notwithstanding.
2	Assistant, reorganise my diary on the fifth so that the important meetings come first.	Sets the importance of the first low-priority meeting to "high" and all other events to "normal", without any further updates.
3	Assistant, is there a time in August where everyone from finance is off?	Returns `True` for the first employee whose vacation starts in August.
4	Assistant, book a conference room for the meeting with sales tomorrow at 2 PM.	Assumes the user is part of the sales team, scheduling a meeting with the wrong attendees as a result.
5	Assistant, add bi-weekly mentorship sessions with the reports of my reports starting next Monday at 2 PM to my calendar.	Hallucinates an end date for the recurrent event, scheduling instances only for six months.
2	Assistant, add a reminder 1 hour before all important meetings, with the meeting title in the subject.	Disregards `add_event` documentation according to which the user should not be a member of attendees lists for events in their own calendar.
7	Assistant, schedule by-monthly team training sessions on the first Monday at 10 am for hires who joined since the 1st of May, alternating between the Engineering and Sales and Marketing.	Cannot correctly resolve the meeting start dates scheduling two meetings which start at the same time in the first Monday of the current month, which has already passed.
8	Assistant, cancel all my meetings Wednesday next week and mark me out of office	Cancels meetings on Wednesday in the current week instead
9	Assistant, how many employees called John are in my team?	Exact matches the `name` attribute instead of calling `find_employee('John')` `and filtering to ensure returned employees are in user's team`
10	Assistant, what date did Joris and Pete meet last week?	Wrong information provided to the user because the model is looking for a meeting involving Joris and Pete in user's calendar as opposed to checking either Joris' or Pete's calendar.
11	Assistant, reschedule the meetings which overlap with the annual review this afternoon to the same time tomorrow.	Adds copies of overlapping events tomorrow, instead of modifying existing events.
12	Assistant, schedule a 30 mins meeting with Frank from finance at 10 AM in any available meeting room.	Schedules a meeting in the wrong room, choosing the first room returned by the room search API without first checking availability for the entire duration specified by the user.
13	Assistant, can you find a room that can accommodate 20 people for a meeting on Thursday afternoon?	Incorrectly processes serch results, returning rooms that are not available during the stated interval
14	Assistant, who in our team has not booked any vacations yet?	Includes the user in the list of returned names, not expected since the user was asking about other team members, not themselves.
15	Assistant, reschedule all meetings from today to next Monday.	Reschedules all the meetings happening until next Monday to next Monday instead of rescheduling today's meetings.

Table 14: Sample task completion errors for gpt3.5-turbo (rows 1-3), gpt-4o-mini (4 - 6), gpt-4o (7 - 9), o1-mini (10 - 12) and o1 (13 - 15).

E.3 Handback control error examples

Id	Query	Agent action
1	Assistant, find a suitable conference room for a meeting with my team I wanna schedule later today.	Tries to schedule a meeting, handing back control because of incorrect diary checking.
	Error cause: Distracted by irrelevant information. The agent is not required to schedule a meeting, not enough details are provided. Instead, it should have searched for a room that is available and has sufficient capacity to accommodate the user and their team.	
2	Assistant, can you find a time slot in my diary today when I could schedule something with the HR department to discuss my performance review?	Hallucinates a program attempting to find HR team, handing back control because it cannot determine it.
	Error cause: Distracted by irrelevant info. The HR team is not defined in the simulation. The task requires the agent to find a slot in user's diary.	
3	Assistant, schedule our team Christmas party 10 days before Christmas. Should start in the morning and end at 10 PM?	Requires the user to provide an alternative date.
	Error cause: Following policy. The agent follows the instruction *Unless the user explicitly states the date, meetings should not be scheduled on or recur during weekends.* which is irrelevant.	
4	Assistant, schedule a follow-up meeting two weeks after my last one-on-one with my manager.	Hand back control because it cannot find the 1:1 meeting.
	Error cause: Documentation comprehension. The agent fails to follow a note according to which the user should not be specified as an attendee during search by convention. The note is included in `find_events` docs and referenced in `find_past_event` documentation.	
5	Assistant, move back my meeting with John from sales and Jane by one hour.	Hands back control because it determines two employees named John are part of the sales team.
	Error cause: Unwarranted disambiguation. The event can uniquely determined by checking the calendar.	

Table 15: Examples of queries where o1 mistakenly hands back control to the user.

E.4 Problem categories

In Table 7, we report task success for five problem categories. Table 16 lists the queries which were used to estimate the performance per problem category. For each query, a model predicts three AEPs with different random seeds, so 30 task completion outcomes are considered when estimating subset performance.

Simple
Assistant, how many meetings with Jianpeng are in my calendar at the moment?
Assistant, plan a weekend trip to the beach with my work colleagues Alice and Bob starting Saturday morning.
Assistant, schedule lunch with my entire team tomorrow at noon.
Assistant, schedule a 3-hour workshop with my team next Monday starting at 1 PM.
Assistant, schedule a meeting with my manager at lunch tomorrow.
Assistant, schedule an urgent meeting with my manager now.
Assistant, share my calendar with my assistant.
Assistant, cancel everything but the important meetings.
Assistant, schedule a team event next Tuesday at 4 PM for 2 hours at the bowling alley.
Assistant, cancel my meeting with Pete and move my meeting with Jianpeng in that slot instead
Constrained scheduling
Assistant, schedule a project update meeting with my manager when I'm free, before 3 PM tomorrow.
Assistant, schedule a project update meeting with my manager when we're both free, before 3 PM tomorrow.
Assistant, set a 3 to 4 meeting in room z with any team members available then.
Assistant, set a 30 mins meeting with Jianpeng at the earliest time when we are both free today.
Assistant, reschedule today's meetings to Monday - keep the same time. If you detect clashes the rescheduled meetings should start as soon as possible after the end of existing events. No overlaps!
Assistant, find an available slot for a 30-minute meeting with my team two weeks from now.
Assistant, is it possible to schedule a team meeting tomorrow 10 am to 11:30 am or is any colleague from my team busy?
Assistant, check my boss' calendar Wednesday to Friday next week, are they available for a meeting?
Assistant, set up a status update meeting with my manager every last Friday of the month at 2 PM till the end of the year. Skip the ones on his holidays.
Assistant, my manager just told me of a clash with our 1:1 tomorrow, reschedule it to the latest free slot we're available.
Complex time expressions
Assistant, show me the last time I met with Alice.
Assistant, schedule a 45-minute team follow-up call two weeks after tomorrow's project deadline, keeping the start time.
Assistant, schedule our team Christmas party 10 days before Christmas. Should start in the morning and end at 10 PM.
Assistant, schedule a 1-hour meeting with my manager, then a 45-minute meeting with my team, followed by a 30-minute meeting with the sales team. Add a 15-minute buffer between each meeting starting tomorrow at 9 AM.
Assistant, put 45 minutes in the calendar, back-to-back, with Engineering and Marketing starting at 10 AM tomorrow... Actually, add a 10-minute buffer between each meeting.
Assistant, find an available conference room for my next meeting and schedule it there.
Assistant, book me out of office for the last two hours of the working day the day before my vacation in October.
Assistant, schedule a 1-hour review meeting with my sales team next Monday at 10, then one with finance right after that, and one with engineering after a 30 mins break.
Assistant, block the last hour of the working day for a catch-up with my team the day before any of their vacations start.
Assistant, change our weekly team meeting to happen on Thursday instead, with a update to say 'friday is a no-meeting day'?
Policy / instruction following
Assistant, schedule a meeting with my team every day next week at 3 PM.
Assistant, plan an off-site event with my team this weekend at Central Park starting at 10 AM.
Assistant, schedule lunch with a different team member each day next week at 12:30 PM.
Assistant, block 90 mins of focus time every morning at 8 AM for the next two weeks.
Assistant, I've got an urgent task that needs 3 hours starting at 1 PM tomorrow. Reschedule my existing meetings to fit this in, but try to keep the same day.
Assistant, schedule a meeting with my team late afternoon tomorrow. Mark Alice optional.
Assistant, reorganise my diary on the fifth so that the important meetings come first.
Assistant, add a strategy review with the CFO and the COO one week from today at 2:30 PM, for 1 hr.
Assistant, set 30 minutes tomorrow late afternoon with the department heads from engineering, finance and marketing.
Assistant, add a reminder 1 hour before all important meetings, with the meeting title in the subject.
Advanced problem solving
Assistant, find a suitable conference room for a meeting with my team I wanna schedule later today.
Assistant, see if my boss' boss and Jane have accepted my meeting request for tomorrow. If anybody declined, reschedule to take place later but at the earliest available time for everyone, I'm free all day.
Assistant, schedule a meeting in the afternoon with my engineering colleagues, avoiding any engineering management.
Assistant, find an available conference room for my next meeting and schedule it there.
Assistant, block 2 hours of free time for holiday preparation after dinner on the last working day before my next vacation.
Assistant, I will need to schedule an important retrospective sometime next week, how many rooms accommodating between 8 and 12 people do we have?
Assistant, add a finance manager to my meeting with the marketing manager.
Assistant, who in finance is yet to book a holiday this year?
Assistant, Ari and James are on holiday next month, who's out for longer?
Assistant, what's ratio of Diarmuid to Anders holidays from the start of the year till the second of July?

Table 16: Listing of queries for which task success is reported in Table 7.

E.5 Primitive selection

Below, we report primitive selection results broken down for three key ASPERA modules. "Task success" represents the task success rate for queries whose sample solution made use of the primitive in question. The final row shows the global precision, global recall, micro F1 and mean task success across primitives in the module.

`work_calendar`

Primitive	Precision	Recall	F1	CCK task success (1-shot)
find_past_events	0.83	0.91	0.87	0.73
RepetitionSpec	0.76	0.94	0.84	0.68
find_events	0.97	0.71	0.82	0.73
summarise_calendar	1.00	0.67	0.80	0.67
get_default_preparation_time	0.67	1.00	0.80	0.00
Event	0.73	0.78	0.76	0.64
delete_event	0.79	0.65	0.71	0.78
find_available_slots	0.73	0.64	0.68	0.76
get_calendar	0.73	0.55	0.63	0.69
add_event	0.97	0.41	0.58	0.66
CalendarSearchSettings	0.29	0.50	0.36	0.75
ShowAsStatus	0.33	0.12	0.18	0.56
get_search_settings	0.33	0.09	0.14	0.73
Overall	0.62	0.61	0.61	0.66

`company_directory`

Primitive	Precision	Recall	F1	CCK task success (1-shot)
get_all_employees	0.98	0.83	0.90	0.71
get_employee_profile	0.87	0.92	0.90	0.71
get_current_user	0.89	0.89	0.89	0.72
Team	0.97	0.79	0.87	0.67
find_reports_of	0.95	0.77	0.85	0.81
find_employee	0.92	0.77	0.84	0.74
find_team_of	0.98	0.68	0.81	0.74
get_vacation_schedule	0.73	0.83	0.77	0.76
get_assistant	1.00	0.60	0.75	1.00
find_manager_of	0.86	0.63	0.73	0.65
Employee	0.05	0.50	0.09	0.75
Overall	0.66	0.74	0.70	0.74

`room_booking`

Primitive	Precision	Recall	F1	CCK task success (1-shot)
room_booking_default_time_window	0.75	1.00	0.86	1.00
find_available_time_slots	0.50	0.50	0.50	0.50
search_conference_room	0.30	0.89	0.45	0.72
summarise_availability	0.06	1.00	0.12	0.67
Overall	0.27	0.42	0.33	0.72

Table 17: Primitive selection results broken down for three ASPERA modules. The final column shows o1's task success in the CCK setting for the subset of queries whose sample solution made use of the primitive in question. This can be thought of as a proxy for how well the model is able to make use of this tool, in contrast to how well it is able to select it.

F Evaluation supplementary material

Given the rapid evolution of LLM capabilities, we additionally evaluate several models from the OpenAI and Gemini families released after our manuscript submission in December 2024. Table 18 summarizes the task success rates achieved by these models in the CCK setting.

OpenAI The GPT-4o release evaluated at the time of submission (May 2025) exhibits a 6.67% increase in task success compared to the variant we assessed in this paper (September 2024), matching the performance of o1-mini. Meanwhile, the o3-mini model demonstrates a substantial 12.27% improvement in task success over the latest GPT-4o, attributed to its superior reasoning capabilities. However, notably, our evaluation shows that o3 underperforms compared to o1 by a 5.06% margin.

The high execution error rates of o3 and o3-mini prompted further analysis of their generated AEPs. This revealed that a considerable number of errors stemmed from improper module imports. Specifically, ASPERA imports were mistakenly included by these models despite explicit instructions (as detailed in Figure 19b) to only import standard library modules, as necessary, because ASPERA-specific imports are automatically handled at runtime. Additionally, o3 occasionally introduced unnecessary future imports (e.g., `from __future__ import annotations`), even after the prompt explicitly specified the Python version for AEP implementation. This behaviour did not changed when the AEP generation prompt was updated to include the `python` version the code should target.

While these errors indicate diminished instruction-following capabilities and redundant code generation tendencies, the primary purpose of *Asper-Bench* is to evaluate an agent's capability to perform complex, multi-step reasoning tasks. Upon manually correcting import-related errors and re-executing the programs, task success increased by 3.05% for o3-mini and 6.26% for o3. The results show that despite discounting instruction adherence errors, o3 does not demonstrate substantially better performance compared to o1 in executing *Asper-Bench* queries.

Gemini Similar issues with disregarding import handling instructions were observed in the latest Gemini models. For the 2.0-flash model, this behavior resulted in a modest 1.6% difference in task success, whereas for 2.5-flash, the discrepancy was significantly larger at 10.27%.

Overall, our findings indicate improvements in models' abilities to handle complex queries. However, particularly challenging queries that demand advanced reasoning and creative tool use (such as query 3 in Table 2) continue to elude all evaluated models. Systematic analysis of these challenging cases can highlight specific weaknesses, guiding the development of increasingly demanding benchmarks as model capabilities evolve – a direction we leave open for future research.

Model	Checkpoint	Task success (%)	Task success (lenient)		Solution err. rate	Execution err. rate	Handback control err. rate
			ASPERA Imports	Future Imports			
o1	o1-preview-2024-09-12	80.13	—	—	62.23	9.61	28.15
o3	o3-2025-04-16	75.07	77.73	81.33	34.70	40.74	24.56
o3-mini	o3-mini-2025-01-31	64.27	67.30	—	59.52	19.32	21.17
GPT-4o (May 25)	gpt-4o-2024-11-20	52.00	—	—	64.16	21.11	14.74
o1-mini	o1-mini-2024-09-12	51.40	—	—	57.43	16.76	25.81
GPT-4o	gpt-4o-2024-05-13	45.33	—	—	49.75	38.79	11.46
GPT-4o-mini	gpt-4o-mini-2024-07-18	21.07	—	—	49.82	42.91	7.27
gpt-3.5-turbo	gpt-3.5-turbo-0125	10.80	—	—	34.23	2.96	62.81
2.5-flash	gemini-2.5-flash-preview-05-20	59.33	69.60	—	39.02	49.83	11.15
2.0-flash	gemini-2.0-flash-001	50.67	52.27	—	70.81	17.57	11.62
1.5-pro	gemini-1.5-pro-002	33.73	—	—	53.54	39.22	7.24
1.5-flash	gemini-1.5-flash-002	27.87	—	—	46.23	45.83	7.95
1.0-pro	gemini-1.0-pro-002	12.67	—	—	27.49	65.49	7.02

Table 18: CCK task-success evaluation for OpenAI and Gemini model families. "—" indicates import errors do not affect these models. **Shaded rows** repeat results reported in Table 3 and Figure 4 to facilitate comparisons.

G Comparison with other benchmarks

This appendix extends §7, providing further comparison between *Asper-Bench* and existing benchmarks in tool use (§G.1), LLM agent evaluation (§G.2), and code generation (§G.3). While our focus is on *Asper-Bench*, it is important to note that developers can extend ASPERA to new domains, and use our data generation engine to create high-quality datasets alongside robust evaluation programs—a key contribution of our work.

G.1 Multiple tool use datasets

Evaluating complex action execution in digital assistants requires datasets grounded in realistic, multi-step queries that integrate multiple tools (§1). Several benchmarks focus on multi-tool queries, but each has limitations which make them unsuitable for assessing complex action execution capabilities in digital assistants. We consider popular datasets, referring the reader to Qu et al. (2024) for an in-depth review of tool-use datasets.

ToolAlpaca Tang et al. (2023) seed ChatGPT3.5 with crawled API names and intended use information to synthesise documentation along with user queries, agent actions and simulated environment response. The resulting instructions primarily involve a small number of API calls (Figure 22), making them better suited for evaluating argument parsing rather than complex multi-tool reasoning. The quality of queries deteriorates as more tools are incorporated. As observed in prior work (Iskander et al., 2024), API call annotations frequently contain hallucinations or missing arguments, limiting the dataset's suitability for our setting[16] .

ToolBench (Qin et al., 2024) improves upon ToolAlpaca by incorporating real-world API documentation instead of synthesising it. While this allows for more diverse tool-use scenarios, multi-tool queries remain sparse in the evaluation set (Figure 22), restricting opportunities to assess complex tool interactions.

Our ToolBench analysis revealed that direct synthesis using real-world API documentation affects query naturalness in several ways. For example, the API names are directly mentioned in the query (Table 19, row 3), a problem which increasingly affects coherence as the number of APIs invoked in the query increases(Table 19, row 4). This arises because the sampled APIs are designed for a wide variety of high-level tasks (e.g., video download, web crawling, weather report, etc) amongst which relationships are sparse and which cannot be naturally combined to synthesise natural complex tasks. Iskander et al. (2024) conduct an in-depth study focused on the impact of query synthesis from API documentations on query quality.

TaskBench Shen et al. (2023) study LLMs for task automation. The authors recognise that task complexity is not only dependent on the number of

[16] In Table 19, row 2, the user specifies the "SalesDB" connection string has changed and has to be modified, but does not specify the new value.

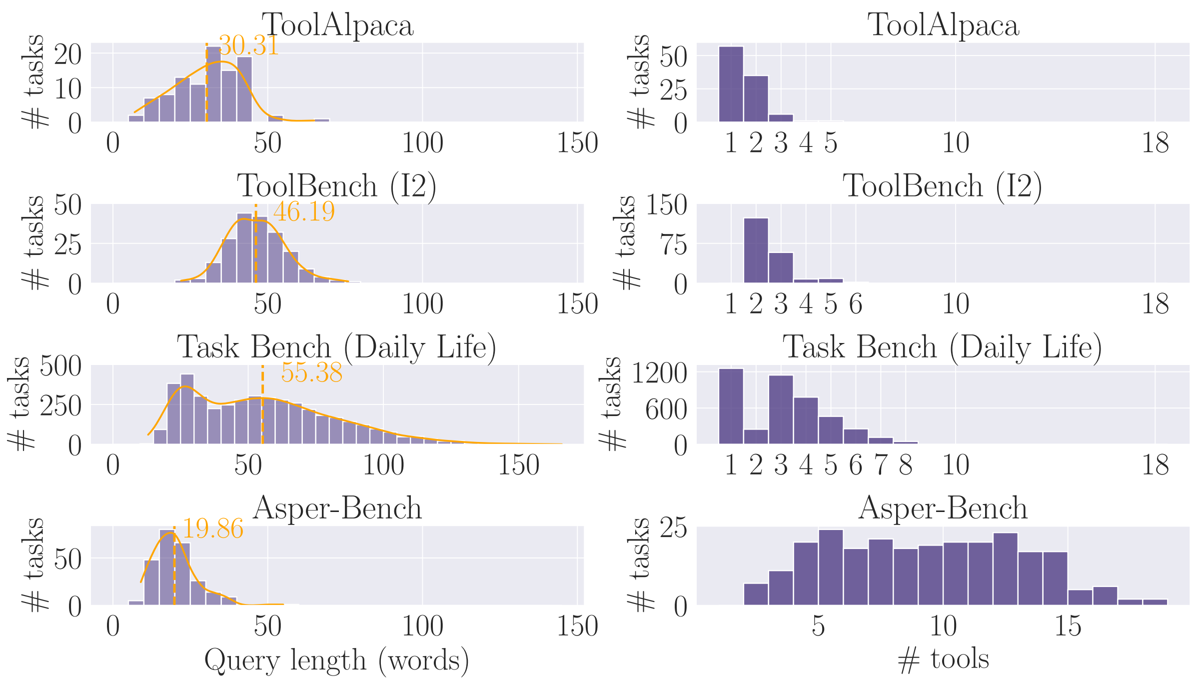


Figure 22: Comparison of query length and action distribution between ToolAlpaca (Tang et al., 2023), ToolBench (Qin et al., 2024), TaskBench (Shen et al., 2023) and *Asper-Bench* (generated with the ASPERA framework, §3).

ID	Number of tools	Corpus	Query
1	1	ToolAlpaca	I'm sending a package to my friend in New York, but I'm not sure if I have the correct address. Can you check if this address is valid and deliverable? Here's the address: 123 Main St, Apt 4B, New York, NY, 10001.
2	3		There's an update to our data source, the connection string has changed. Can you modify the existing data source called "SalesDB"? After you've done that, I'd also like to add a new chart to our "Sales Overview" dashboard, called "Top Selling Products". Don't forget to update the dashboard name to "Complete Sales Overview".
3	2	ToolBench	I'm working on a personal project and I need to gather a large number of random anime images. Can you provide me with around 5000 random anime images from the Random anime img API? Additionally, I would like to create profile images for my project using the Image Service API.
4	3		I'm planning a family road trip and I want to create a playlist of MP3 songs. Can you convert the audio from the YouTube videos with the ids 'UxxajLWwzqY', 'abc123', 'xyz456' to MP3 format? Please provide the download links and make sure the converted files are free of any profanity.
8	3	TaskBench (daily life)	I'm enrolled in a Data Science Conference happening on May 15, 2023. Could you help me manage the logistics? Let's start by scheduling a flight from Los Angeles to New York for the conference day and ensure I'm reminded of the meeting at 2 PM.
9	7		I'm planning on applying for a software development job soon and I also need to purchase a new Smartphone from Amazon for my everyday tasks. Could you assist me with these tasks? I also need to prepare for the interview, so I would appreciate it if you could help me record notes on a few topics such as data structures, problem solving, and algorithm design. Plus, I want to record an audio file named 'example.wav'. After purchasing the Smartphone, could you make sure it is delivered to my home address and send me an SMS on 1234567890 to confirm its arrival? By the way, could you install the Zoom application on my computer to facilitate video conferencing.

Table 19: Samples from multiple tool use corpora.

tools, but also on the relationships between them, which the documentation- and template-based synthesis approaches of Qin et al. (2024) and Tang et al. (2023) do not model. To address this, they ground query generation in a graph which encodes tool dependencies. In their framework, single-node graphs model simple tasks and more general chain and directed-acyclic graphs (DAGs) ground tasks

Benchmark	Dynamic Interaction?	Realistic Environment?	Diverse Human Tasks?	Functional Correctness?
Mind2Web (Deng et al., 2023)	✗	✓	✓	✗
Form/QAWeb (Shi et al., 2017)	✗	✓	✓	✗
MiniWoB++ (Liu et al., 2018)	✓	✗	✗	✓
WebShop (Yao et al., 2022)	✓	✗	✗	✓
ALFRED (Shridhar et al., 2020)	✓	✗	✗	✓
VirtualHome (Puig et al., 2018)	✗	✗	✓	✗
AndroidEnv (Toyama et al., 2021)	✓	✓	✗	✗
WebArena (Zhou et al., 2024)	✓	✓	✓	✓
WorkflowLLM (Fan et al., 2024)	✗	✗	✗	✗
OfficeBench (Fan et al., 2024)	✓	N/A	✓	✓
Asper-Bench (ours)	✓	N/A	✓	✓

Table 20: Comparison of *Asper-Bench* with agent benchmarks.

with higher complexity. Table 19 shows an example of a query grounded in DAG tool graph (row 8). These are more complex and natural compared to ToolBench queries as API parameters are shared across tasks, and, more generally, API inputs can depend on the output of previous tasks. However, such relationships become sparse at the number of tools increases, and, as result, queries grounded in chain graphs with multiple nodes (row 9) are unnatural and pose limited additional challenges to LLMs compared to parsing single API calls.

G.2 LLM agent benchmarks

A growing body of research focuses on developing autonomous LLM-based agents, primarily for web-based tasks (Zhou et al., 2024; Shi et al., 2017; Liu et al., 2018; Yao et al., 2022; Deng et al., 2023; Shridhar et al., 2020), home automation (Puig et al., 2018), mobile development (Toyama et al., 2021), open-ended computer tasks (Xie et al., 2024), and workplace assistance (Fan et al., 2024; Wang et al., 2024b). Some of these environments are designed to support reinforcement learning research (Toyama et al., 2021), while others benchmark visually grounded agents (Shridhar et al., 2020; Puig et al., 2018). In contrast, *Asper-Bench* targets digital assistant capabilities (e.g., Alexa, Siri), emphasising the parsing of complex user actions into executable programs rather than long-horizon planning, a central theme in most LLM agent benchmarks.

One key distinction between *Asper-Bench* and existing agent benchmarks lies in action space complexity. Web-based agent benchmarks typically define small, discrete action sets; for example, WebArena (Zhou et al., 2024) includes just 12 actions, with simple textual descriptions such as `new_tab` (*Open a new tab*). In contrast, *Asper-Bench* features 69 actions, including both high-level commands like `delete_event` and low-level primitives such as `get_next_dow`[17]. This richer action space supports nuanced execution, requiring reasoning over fine-grained dependencies instead of following step-by-step workflows.. As such *Asper-Bench* is a code generation benchmark which tests language understanding, logical reasoning and short-term planning capability *when the agents are grounded in fine-grained dependencies, unseen during pre-training* whereas other benchmarks provide a complementary view of LLMs' long-term planning capability.

Table 20 compares *Asper-Bench* with other benchmarks across four key dimensions: whether the environment allows dynamic interaction, whether tasks are grounded in realistic environments, whether datasets include diverse human-verified tasks, and whether the benchmark ensures functional correctness through execution. Unlike most of these, ASPERA supports dynamic interaction[18] and functional correctness evaluation. Moreover, *Asper-Bench* is a diverse collection of human-verified tasks. ASPERA uniquely enables developers to generate high-quality tasks and evaluation code through LLM interaction to support benchmarking on custom use cases.

The simulation fidelity depends on task complexity and is more readily achieved for web benchmarks which rely on widely used open-source technologies. In contrast, more complex environments such as OfficeBench (Wang et al., 2024b) and digital assistants are difficult to simulate since they rely on proprietary, commercial technologies. In ASPERA, we tackle this by implementing a simplified but fine-grained `python` simulation of a fictitious corporate calendar-management application which supports our objective of evaluating LLMs' capability of complex action execution via program synthesis.

G.3 Other code generation benchmarks

As discussed in §7, *Asper-Bench* is a code-generation benchmark which tests LLMs complex action execution capability given *custom, project-runnable dependencies*. This significantly more challenging setting is uncommon, as it requires a custom simulation environment (Siddiq

[17] A function for computing the next occurrence of a specified weekday.

[18] For example, the agents have access to the stack trace. See `src/aspera/evaluator.py::get_solution_feedback` in our code release.

Benchmark	**Spearman r**	**Pearson p**
EvalPlus (Liu et al., 2023)	0.8909	0.8909
BigCode (Hard) (Zhuo et al., 2024)	0.9879	0.9879

Table 21: Correlation between model ranks on ASPERA with other standard code generation benchmarks. Scores for Gemini 1.5-Flash and 1.0-Pro are not reported on the BigCode (Hard) leaderboard and EvalPlus leaderboard does not include CodeGemma (27B). Hence, we exclude these models from the analyses.

et al., 2024). In contrast, function generation focusing on competitive programming (Liu et al., 2023) requires only standard library dependencies, whereas more general software capability benchmarks (Zhuo et al., 2024) assess program generation based on widely used dependencies seen in training (e.g., `numpy` apis). Consequently, *Asper-Bench* complements existing benchmarks by assessing program generation under custom dependencies. While prior benchmarks (e.g., (Liu et al., 2023)) are increasingly saturated by strong LLMs, §5 shows *Asper-Bench* remains challenging. However, *Asper-Bench* performance correlates with model ability on popular LLM code benchmarks (Table 21).