No Debug, we see that automated debugging generally improves performance dramatically. Figure 3 shows that even one step of automated debugging helps substantially, and further steps exhibit diminishing marginal improvements. Table 2 reports the fraction of error types encountered during training across. Python exceptions are most common, followed closely by errors in plan semantics, then errors in plan syntax, and finally timeouts. We also see that the error types are well-distributed within successful trials, suggesting that each of the four types of feedback are beneficial. In general, GPT-4 tends to make small, local corrections to code during automated debugging. If the code is structurally flawed and requires a significant rewrite, restarting the dialogue from the beginning may be required. **The role of PDDL names.** Examining the results for the No Names ablation, we see performance overall is very poor. This confirms our hypothesis that the terms present in the PDDL domains and tasks are helpful to the LLM, as they would be to a human. Note that planners like Fast Downward and generalized planners like PG3 would be unaffected by name changes. However, there are a few cases where the No Names ablation does succeed, suggesting that the LLM has some capacity for purely syntactic generalized planning.

GPT-3.5 vs. GPT-4. Examining the results for GPT-3.5, we see that it performs much worse than GPT-4. This is consistent with other reports (**??**) that GPT-4 is far superior on reasoning and coding tasks. Qualitatively, the programs proposed by GPT-3.5 are flawed in myriad ways and do not usually appear "close". They also do not seem to improve with automated debugging. **Data efficiency.** In the appendix, we analyze the number of training tasks *used* in each successful trial. A training task is used if it appeared in the prompt and/or triggered feedback during automated debugging. Since two training tasks are always used in the prompt, the minimum used is two. Interestingly, in the vast majority of cases, only those two training tasks are used. During automated debugging, these two prompting tasks are always checked first, and most of the time, they are sufficient to identify issues. In a small number of cases, a third task is also used during automated debugging. This result speaks to the strong few-shot learning capabilities of GPT-4. We expect that in many cases, even one training task would suffice, although we did witness a drop in performance in preliminary experiments with one task.

Discussion and Future Work

In this work, we showed that GPT-4 with CoT summarization and automated debugging is a surprisingly strong generalized planner in PDDL domains. We conclude with limitations of this work, reflections about the implications of our findings, and opportunities for future work.

Limitations. A major limitation of this work and previous work on generalized planning is that it is easy enough to hand-design generalized plans for all of the domains considered. Nonetheless, we expect this line of work to be practically useful for at least three reasons. (1) In some cases, it may be considerably easier to specify PDDL domain and problem descriptions than it is to directly specify a generalized plan. (2) In a fully autonomous system, where operators and predicates are learned *in association to natural language*, we would want the system to also synthesize generalized plans autonomously. (3) Beyond PDDL, generalized planning with LLMs would be an even more attractive option, since other approaches rely strongly on formal specifications. Another limitation of this work is our use of training tasks to communicate the task distribution of interest to the LLM. In general, a few example tasks may be insufficient to express the full distribution. Other representations like natural language or procedural generation code may be better, but would require more human input.

Is (generalized) planning now obsolete? No. First, there remains a performance gap between GPT-4 and PG3, and other generalized planners may be even better. However, even if this gap is closed by the next generation of LLMs, we would still say no. *Planning* remains essential in domains where no simple program exists. An interesting direction for future work would be automatically detecting whether a simple program might exist before attempting to synthesize one. We tried the Sokoban domain and found that GPT-4 correctly indicates that no simple program exists. However, this property of Sokoban is well-known, so it is likely parroting pretraining data. We also tried the Slitherlink domain, which was featured in the 2023 International Planning Competition, and found that GPT-4 did *not* recognize that no simple strategy exists (**?**). *Generalized* planning without LLMs also remains important in cases where domain descriptions are not human-readable, e.g., because the predicates or operators are learned (**?**). Even with natural language descriptions, combining "classical" approaches with LLMs may be best.

What if we gave the LLM access to a planner? Giving an LLM access to APIs is a very powerful idea (**?**) and one such API could be a PDDL planner (**?**). An LLM could potentially use such a planner for generalized planning, especially given that approaches like PG3 rely on access to a planner to generate example plans. In some domains, generating example plans naively would likely confuse the LLM. For example, plans generated in the Forest domain would follow arbitrary paths through the dirt rather than following the slightly longer marked trail. In other cases, though, example plans could be very useful, especially if the LLM generates them in a targeted way. Leveraging *diverse* plans (**??**) could be particularly useful.

Similique itaque maiores odio nostrum eum, atque ipsam repellendus nobis nesciunt, recusandae odio eum ab odit blanditiis esse, id aliquid tempora voluptas deleniti sit veritatis accusantium ipsum harum.Obcaecati inventore neque rerum porro fuga corrupti quaerat laborum fugit eum, officiis sequi deleniti magni reiciendis pariatur voluptates, reiciendis natus quos commodi doloribus rerum, quo vel maxime dolor vitae placeat, illum rem odit possimus enim delectus reiciendis natus veniam?Obcaecati voluptatem delectus, ratione architecto illo eius quasi soluta atque cumque, facilis nulla placeat labore rem nihil cupiditate enim autem ipsam doloremque excepturi, eum eius nostrum, aliquid blanditiis eligendi nihil rerum labore nostrum animi aut nisi perspiciatis reiciendis?Praesentium eius eaque nostrum at fuga sint asperiores beatae recusandae, cum ad hic

odit iste tempora magni sed aperiam, dolor cupiditate cum voluptatem.Ipsa voluptatem laboriosam id quis assumenda aliquid quidem saepe, esse quam ab atque