Synthetic Function Demonstrations Improve Generation in Low-Resource Programming Languages

Nick McKenna[†1], **Xinnuo Xu**[†], **Jack Williams**[†], **Nick Wilson**[†]
Benjamin Van Durme[‡], **Christian Poelitz**[†]
[†]Microsoft Research, [‡]Microsoft
nmckenna@microsoft.com, cpoelitz@microsoft.com

Abstract

A key consideration when training an LLM is whether the target language is more or less resourced, for example English compared to Welsh, or Python compared to Excel. Typical training data for programming languages consists of real program demonstrations coupled with explanatory human-written comments. In this work we present a novel approach to the creation of such data for low resource programming languages, which lack naturally occurring data. Our process generates synthetic, textbook-quality demonstrations of how to use library functions, which we show makes for good model finetuning data. We demonstrate in an example domain of Excel Formulas. First, we collate language documentation, then we use this to augment a powerful teacher model which generates synthetic training data, and finally finetune student models on the demonstrations. Our technique improves student performance on 2 question-answering datasets: WikiTQ and TAT-QA. We also show advantages of finetuning over standard RAG approaches, which can offer only modest improvement due to the unfamiliarity of the target domain to student models.

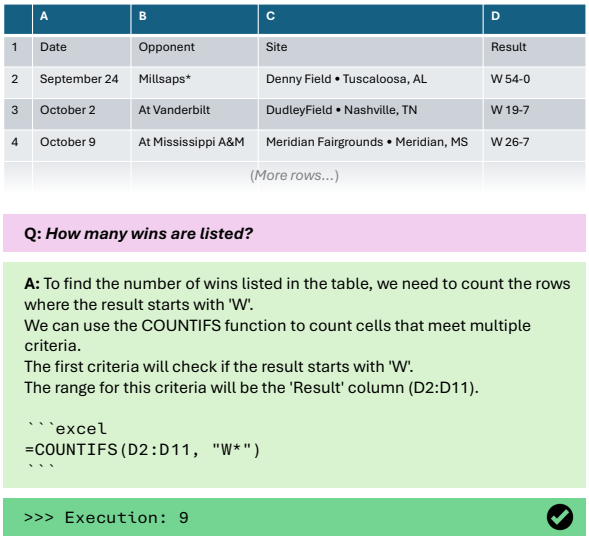
Keywords: Conversational Systems, Language Modelling, Less-Resourced Languages, Automatic Generation of Training Data, Fine-tuning, Adaptation, Validation of Language Resources

1. Introduction

The process of pretraining LLMs on code has improved significantly, supporting high accuracy single-shot generation for general-purpose programming languages (PLs) like Python, when prompted with a variety of natural- and programming language inputs (OpenAI et al., 2024b; Abdin et al., 2024; Yang et al., 2025). These high-resource PLs (HRPLs) profit due to the abundance of training data, which is often human-annotated with explanatory comments (Kocetkov et al., 2022; Lozhkov et al., 2024). Comparatively, LLM generations in Low-Resource Programming Languages (LRPLs) are of significantly worse quality due to the limited data available to train models (Joel et al., 2024). In this paper, we show that *synthetically generated* data can be used to improve performance when generating in an LRPL. [2]

Previous studies directly augment real-world data in high-resource languages using SoTA LLMs (Honovich et al., 2023; Yoo et al., 2021). However, by definition even SoTA LLMs such as GPT-4o struggle to generate code in low-resource languages (Joel et al., 2024), making it challenging to directly generate new training data. This makes such previous methods, and more general knowledge distillation approaches, less appropriate for LRPLs.

Fortunately, many LRPLs have public, human-written *documentation* detailing language specifications and core library function usage. In this work, we instead generate textbook-quality demonstrations based on this documentation (Gunasekar et al., 2023). Generating demonstrations that span the vast space of possible programs is also a challenging task. However, modern LLMs have already learned basic principals of programming and problem-solving by training on plentiful data from



	A	B	C	D
1	Date	Opponent	Site	Result
2	September 24	Millsaps*	Denny Field • Tuscaloosa, AL	W 54-0
3	October 2	At Vanderbilt	DudleyField • Nashville, TN	W 19-7
4	October 9	At Mississippi A&M	Meridian Fairgrounds • Meridian, MS	W 26-7
		(More rows...)		

Figure 1: Finetuning on synthetic data improves adaptation to the low-resource domain. Qwen2.5-coder 3B learns to read the table and compose a formula with flexible string formatting, executing to the answer.

[1]Now at GitHub Applied Science.

[2]Code, prompts, and data are available at: https://github.com/microsoft/Synthetic-Function-Demonstrations.

Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), pages 3094–3106
11–16 May 2026.

HRPLs, and we hypothesize that these concepts will transfer when adapting to a new programming language domain. We instead restrict the scope of generated data to the target LRPL's standard library, generating rich demonstrations of each function.

Our goal is to develop data for enhancing model generations which (1) efficiently captures the syntax and semantics of the target LRPL using a minimal amount of data, and (2) leverages models' existing knowledge of general programming concepts, learned from pretraining on large HRPL data, to reason through complex problems in the target domain using the new functions.

Data sparsity is a difficult problem, especially for LRPLs like niche Domain-Specific Languages (DSLs). In this paper we focus on Excel Formulas (Zhao et al., 2024): though widely used by millions, formulas are considered a low resource language because of the lack of large-scale natural data sufficient for LLM training (Kocetkov et al., 2022). We examine the task of question-answering on tabular data through augmentation strategies on student models to improve generation in Excel formulas.

This work primarily proposes a general protocol for efficiently generating synthetic training data for a target LRPL. The data robustly demonstrates the usage of library functions in natural contexts and with assistive explanations, tuned to improve model performance on a task. We demonstrate efficacy of the approach by augmenting teacher model GPT-4o (which we show is not itself Excel-specialized), and two student LLM model families, Qwen 2.5 and Llama 2 (chosen for their open weights and diverse family of pretrained models) on two table-based QA datasets, WikiTQ and TAT-QA. Within each model family, we also test the effects of model scale, and also dedicated code-pretraining vs. not.

Our contributions are as follows:

1. We define a protocol for synthesizing domain-targeted training data to teach an LRPL to student models. Generation is done by augmenting a teacher LLM *without* strong understanding of the target domain simply by retrieving rich function documentation and sampling a general data context for realistic grounding.

2. We show that finetuning on our synthetic examples improves table-QA accuracy by more than 10% for most student models. Also, we show that code pre-training in HRPLs does not directly improve Excel generation, but rather makes domain adaptation through finetuning on Excel more effective.

3. We show that finetuning is necessary in such an unfamiliar LRPL as Excel by comparing with typical RAG-only techniques, which do not significantly improve generation accuracy, even with access to oracle-retrieved information.

2. Background

Code has become integral to general LLM pretraining data and techniques (DeepSeek-AI et al., 2025), and to logical reasoning at test-time through code generation (Madaan et al., 2022; Puerto et al., 2024) and tool invocation (Schick et al., 2023).

Like natural languages, programming languages suffer from a large disparity in resources (Maguerresse et al., 2020; Lozhkov et al., 2024): generation in low-resource languages is demonstrably worse than in high-resource ones, even by the same model (Joel et al., 2024). In particular, Excel formula language is a unique LRPL because it is used by so many people, yet there exists little natural training data for formulas. This results in poor generation performance even from top models, which we also document in this work (§4.1).

While Payan et al. (2023) produce an Excel-specific benchmark which could be used for training or testing, it contains mostly OfficeScript, a different Excel language distinct from Formulas. In contrast, this work presents a method for generating arbitrary amounts of synthetic data tuned to a target domain, and we demonstrate this for Excel formulas. Another study, Zhao et al. (2024), converts SQL queries (which are abundant) into Excel ones, but this data is unnatural and biased to SQL capabilities and use-cases. In this work we generate synthetic data arising from in-domain documentation and spanning the native capabilities that are described there.

3. Generating Synthetic Training Data

Due to the lack of pretraining data, even SoTA LLMs' performance is unreliable when generating in an LRPL (Joel et al., 2024). As we show in §4.1, even GPT-4o scores only 58.2% on a key Excel test set, showing this challenge. We must thus develop our method without requiring strong performance in the target domain by the teacher model. Fortunately, Excel (and other LRPLs) have rich, public documentation of language specifications and library functions. We leverage this (along with verification methods) to augment the teacher model and generate high-quality function demonstrations for the student model. We hypothesize that the student will learn to apply basic Excel functionality while transferring general problem-solving skills that it learned from pretraining in HRPLs.

Our pipeline only requires (a) LRPL function documentation, (b) unannotated, natural data contexts to serve as grounding, and (c) a teacher model which is generally strong at instruction-following (without requiring strong skills in the target LRPL domain). We demonstrate the method with Excel formulas, though it is suitable for any LRPL with

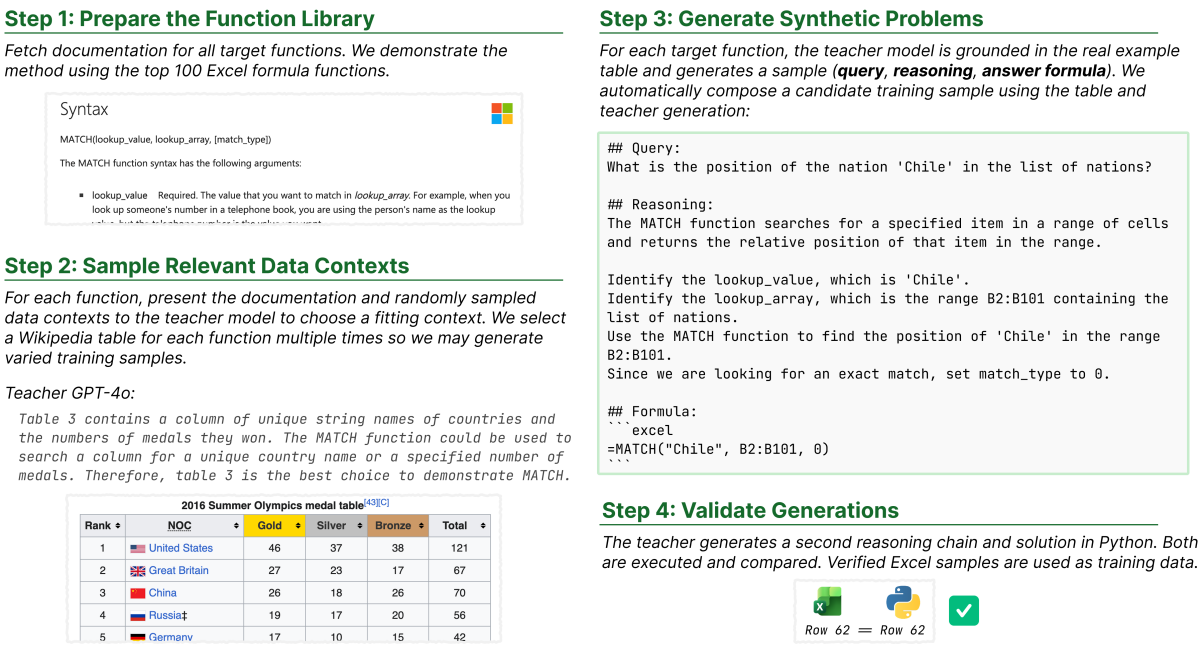


Figure 2: Our pipeline generates textbook-quality synthetic examples which demonstrate how to use target library functions. Examples are grounded in sampled, real data contexts, so the teacher model generates more creative problems and the student model learns from rich, varied demonstrations. We demonstrate producing a synthetic training example for the Excel MATCH function.

these basic resources. It consists of 4 steps:

3.1. Prepare the Function Library

We first collect a library of target functions and their documentation. Excel Formula language contains 505 functions (Microsoft Support, 2024), yet many are unused due to niche functionality, or being superseded by newer implementations[3]. We next gather a small sample of public, unlabeled spreadsheets from the web, similarly to Fisher and Rothermel (2005), and estimate a rough distribution of real function usage. We take the 100 most frequent functions and collect documentation pages from public Microsoft support.

For the function f_i the documentation page d_i contains a brief description of its purpose, its argument vector $\mathbf{a}_i$ (including optional arguments), and their types. d_i often also includes example usages of f_i on a toy data table (a $\sim$5x5 table).

3.2. Sample Relevant Data Contexts

(Zhang et al., 2025) show that grounding synthetic generation in realistic data contexts aids model creativity and alignment to the target format. For simplicity and to demonstrate generality of the method, we seek easily accessible data. For Excel, WikiTableQuestions (Pasupat and Liang, 2015), contains real data tables from Wikipedia; for other LRPLs different scenario data can be chosen. We randomly sample 10 Wikipedia tables per function from the training split. We query our teacher model GPT-4o to pick the best scenario table for each function, such that the function may be executable on the table. Manual inspection shows good quality: GPT-4o matches numeric functions to tables with numeric data, string functions to those with string data, etc.

Method details: In the prompt to GPT-4o we insert the function f_i, its documentation page d_i, and 10 randomly selected tables.

In one sampling batch we acquire a random table for each of the 100 most-used functions in our target library and proceed to the next step with 100 (function, table) tuples. We repeat our entire pipeline until we have generated a suitable-sized curriculum; student models will thus train on a given function f_i across many randomly sampled tables, to aid generalization of f_i to new contexts.

3.3. Generate Synthetic Problems

We aim to generate synthetic demonstrations of function semantics by designing synthetic data samples which systematically show the effect of one function argument at a time. We query GPT-4o (denoted $Q(\cdot)$) to produce multiple synthetic problems from each input by demonstrating the target

[3]Such as the newer XLOOKUP function which supersedes VLOOKUP and HLOOKUP.

function f_i with *each* of its arguments in $\mathbf{a}_i$ found in documentation d_i. All generations are grounded on the chosen table t_i:

$$Q(f_i, d_i, t_i) \rightarrow S_i$$
$$S_i = \{(q, e, a)_j\}_{j \in 1..|\mathbf{a}_i|}$$

Even the teacher model may not be an expert in the LRPL, so d_i provides the semantics of f_i, and the table t_i provides grounding to generate natural questions. For one input, teacher GPT-4o is instructed to produce an output set of sample problems $z_j \in S_i$, where each $z_j = (q, e, a)$. The natural language question q may be solved by executing f_i on t_i while making use of function argument j of f_i. For each problem z_j we also generate a step-by-step explanation e of how to solve the problem, and an executable answer formula a.

Rotating through each function argument and producing one demonstration for each allows us to demonstrate the usage of individual arguments and even optional ones.

Method details: We prompt the teacher model GPT-4o to generate textbook-quality tutorials in a QA format, demonstrating how to use a given function. A key aspect of the prompt is that we instruct the teacher to generate a set of examples, each one demonstrating how to use one argument slot of the target function. This is our method's most detailed and important prompt, so we share it in Figure 3. We elicit tutorial information in JSON, which allows us to extract structured information and compile the training tutorials automatically using a template.

3.4. Validate Generations

Each generated problem demonstrates in textbook-quality the basic function semantics in the target domain, with natural language explanation. However, since samples are synthetically generated using an AI model, they are unverified for correctness. We improve data quality by employing post-generation validation. First, we filter out all samples with answer formula a which fail to execute. Second, we feed generated samples back into the teacher model and generate parallel solutions in Python to the same questions. We do not know the correct answer to synthetic questions without human labeling, but we assume Python (as an HRPL) will have the most reliable generations (Joel et al., 2024). Indeed, we find roughly a 50% match rate between Excel and Python executed values, and retain the verified samples.

We show that this validation is useful. We finetune a Qwen2.5-Coder 14B model on equal amounts of validated and un-validated data (6,440 samples in each) per the procedure in §4.5. We observe that when trained on unvalidated data, the model improves from 46.95% to 52.11% on the downstream WikiTQ test set used in experiments (§4.1), compared to 54.93% when finetuned on an equal amount of validated data. This demonstrates that (a) unvalidated data has great potential even with just simple filtering out of non-executable samples, and (b) if available, parallel generation of solutions in an HRPL like Python can provide useful supervision for higher-quality assurance, resulting in downstream task improvement.

Method details: The teacher model generates sample problems with Excel reasoning and solutions. A problem generated by the teacher model forms a tuple: (table, query, Excel formula, Excel execution) (we execute the formula ourselves for reliability). We format tuples into training data after filtering them for quality. When filtering we generate a parallel solution to each problem in Python, a high resource programming language in which we expect model solutions to be of high quality. We give nearly the same prompt as in model evaluations, but cast each sample as a Python problem using a Pandas DataFrame instead of an Excel spreadsheet. We collect the generated Python code and execute it so that we may compare the Python and Excel executed values.

Because Pandas DataFrames have subtle differences to spreadsheets (e.g. they are 0-indexed whereas Excel is 1-indexed), we develop some basic rules for determining equality between executed values. We are able to accept around 50% of synthetic generations after Python-validation.

4. Experiment: Excel Generation

We augment models in several ways and compare performance in Excel formula generation.

4.1. Example Domain and Datasets

LRPLs lack publicly available data. Excel, while popular, is mainly used in enterprise settings where tabular data is sensitive and private, making it rare to find real-world examples that pair human-written Excel formulas with tables, and natural language queries. Thus, it is an LRPL. Accordingly, we recast existing table-based QA datasets into the Excel domain by embedding tables into Excel spreadsheets and prompting models to answer using Excel formulas:

- **WikiTQ**: WikiTableQuestions (Pasupat and Liang, 2015) contains tables from Wikipedia annotated by humans with questions and answers in natural language.
- **TAT-QA**: A financial table-QA dataset in a similar format as WikiTQ, and in a domain relevant

Teacher Prompt: Generate Examples

```
## General Instruction:
You are a helpful assistant to a data scientist who is learning to use Excel.

You are tasked with creating a tutorial of examples demonstrating the functionality of F, given F's reference documentation, as well as a random data table T taken from Wikipedia.
The tutorial should contain at least one example demonstrating each of F's argument slots, in order to thoroughly describe how F works.

## Task:
First, analyze the documentation of the function F to understand what each argument does. Write a brief explanation of what each argument is used for, including whether it is required or optional.
Format the explanation as markdown:

```markdown
Function: F
- arg1 <required>: explanation of arg1
- arg2 <required>: explanation of arg2
- arg3 <optional>: explanation of arg3
```

Second, write a series of examples demonstrating the use of F on the table T. Each example should contain:

1. The function F
2. The argument A being demonstrated
3. A natural language query Q which requires the use of F and A executed on the table T to compute a solution. Write the query in a natural and realistic way, as if an interested person were trying to analyze the data table to solve a problem.
Make the query specific so there is only one correct answer. For example, to demonstrate a string manipulation function, the query Q should specify exactly how to format the output string so that a program can be written to do this.
4. A brief explanation of what F does in general (not related to the query Q or table T).
5. A step by step explanation of how to use F and A to solve the query Q given T. When explaining the steps, only use values mentioned in the query Q or references into the table T. Use the syntax section of the function F's documentation to explain how the arguments are used.
6. The answer to the query Q. After any reasoning, restate the answer on its own line at the end, e.g. "True", "False", "5", etc.
7. The final Excel formula using F and A to solve the query Q
8. Write the parameter name and required/optional for each of the final arguments given to F as a list, e.g. "param1 <required>", "param2 <optional>", etc.

Write examples which demonstrate the required arguments, then examples for each of the optional arguments.
Format the examples as a JSON list according to the following structure:

```json
(...JSON description omitted for brevity...)
```

For the Excel formula, use the following format:
"=FUNCTION(ARGUMENTS)"

Function:
MATCH

Documentation:
(...MATCH documentation omitted for brevity...)

Random Table:
In Excel tables, the first row is usually reserved for column headers. The first column is usually reserved for row headers. For example, the data starts in A2.
Larger tables may be excerpted here. If so, the first and last rows of the table will be shown, with an ellipsis (...) in between representing the hidden middle rows.
Remember that NaN values in Excel may be written in the table as "nan".

(...table omitted for brevity...)

Tutorials:
```

Figure 3: We instruct teacher GPT-4o to generate multiple tutorials for a target function MATCH, demonstrating each function argument. We add minimal details about Excel solely to describe the data format such as the excerpting process for large tables. The method can be tailored easily to a target LRPL.

to Excel generation (Zhu et al., 2021).

To ensure these problems are solvable using Excel, we use OpenAI GPT-4o and o1 (OpenAI et al., 2024a) to attempt a solution to each problem using Excel, and we collect problems which are solvable in this domain. On WikiTQ cast into Excel, GPT-4o scores 58.2% and o1 scores 67.8%. We cannot know if o1's score being far below 100% is

	Base Model	RAG_{All}	RAG_{Oracle}	FT_{Doc}	FT_{Doc-QA}	FT_{Syn-QA}
GPT-4o	79.19	78.25	**84.19**	-	-	-
Qwen2.5-coder 3B	15.34	13.77	18.94	13.62	15.34	**28.64**
Qwen2.5-coder 14B	46.95	44.76	53.52	40.22	46.95	**54.93**
Qwen2.5 3B	14.87	12.21	17.84	13.62	14.55	**20.81**
Qwen2.5 14B	50.70	47.57	49.45	46.64	49.92	**51.02**
CodeLlama-Instruct 7B	0.47	0.31	0.47	0.78	**9.23**	4.85
CodeLlama-Instruct 13B	10.80	13.15	16.12	12.21	12.83	**21.28**
Llama2 7B	0.63	0.16	1.10	0.63	3.91	**8.61**
Llama2 13B	1.72	0.31	1.56	0.47	4.54	**15.49**

Table 1: Evaluation results on our subset of WikiTableQuestions. Execution Match (EM) measures the percentage of programs which execute to the correct answer. We test (a) base models; (b) RAG settings: all function signatures (RAG_{All}) and oracle-retrieved signatures (RAG_{Oracle}); and (c) finetuned models: using function documentation (FT_{Doc}), and QA-formatted documentation (FT_{Doc-QA}), and synthetic problems (FT_{Syn-QA}). Finetuning on synthetic problems performs better than oracle-retrieved RAG or finetuning on purely documentation-based data.

due to either model incapability in the Excel domain or problem incompatibility when cast into Excel. However, GPT-4o's score below o1's *must* be due to model incapability, demonstrating that it is not strong in our target LRPL domain off-the-shelf.

Therefore, we take the subset of samples passed by the stronger o1 as our test dataset, since they are proven to be solvable in the Excel domain, and we use the o1 formulas as the **oracle** Excel solution for each sample. This results in 639 problems for our **WikiTQ** test and 459 for **TAT-QA** test. In experiments we evaluate model performance using program execution match (EM) to executed values and do not grade the generated formulas; we only use the oracle formulas for a baseline comparison.

4.2. Student Models

We select student model families that are (1) open-weights, (2) available in multiple sizes, and importantly, (3) have corresponding code-finetuned versions of generally trained models.

- **Qwen 2.5** A recent family with notable performance across tasks, including analytical reasoning and coding (Yang et al., 2025). We use Qwen2.5 3B and 14B; and Qwen2.5-Coder 3B and 14B.
- **Llama 2** An older, but popular family with corresponding code-finetuned models (Touvron et al., 2023). We use Llama2 7B and 13B; and CodeLlama 7B and 13B.

4.3. Student Model Augmentation

We compare several student model augmentations which broadly fall into two categories: off-the-shelf RAG methods and finetuning methods.

4.4. RAG

Retrieval-Augmented Generation is a cheap and effective technique to supply novel information to a model by directly inserting textual knowledge from external sources into the model prompt (Lewis et al., 2020). We simulate baselines for both naïve and optimal RAG scenarios in which LRPL function library data is given to the model.

- In the **RAG**$_{All}$ setting, we provide student models with all of the 100 most frequently used Excel function[4] signatures and their descriptions before the question. However, this may naturally create a difficult needle-in-the-haystack scenario where information gets lost.
- **RAG**$_{Oracle}$ is an expected upper bound in which we augment with *only* the functions included in the o1 oracle Excel solution for each question. This simulates an optimal retriever and minimizes the context a student model must consider.

4.5. Fine-tuning

We compare several finetuning scenarios demonstrating common approaches. Our method is:

- **FT**$_{Syn-QA}$: finetuning student models on our synthetic demonstration data.

For **FT**$_{Syn-QA}$ we generate 6,440 validated samples using our pipeline (§3), which demonstrate the top 100 Excel functions and their argument semantics. We then finetune student models on this data. Models train to convergence on a heldout set of 100 questions from WikiTQ dev, typically 1-3 epochs. We finetune all models using the hyperparameters

[4]The same 100 functions as in finetuning experiments.

Hyperparameter	Value	Search Range
Batch size	4	{2, 4, 8, 16, 32}
Learning Rate	5e-5	{1e-5, 5e-5, 1e-4, 5e-4}
LoRA r	64	{32, 64, 128}
LoRA α	1	
Max epochs	6	
Patience	3	
LR scheduler	cosine	{linear, cosine}
Warmup ratio	0.5	

Table 2: Hyperparameter choices for finetuning.

shown in Table 2. We chose these by doing hyperparameter search over the given values using Qwen2.5-coder 3B and choosing the parameters which yielded the highest score on the heldout validation dataset. We then used these settings to train all our models, and applied these finetuned models to the test sets.

We also test two finetuning baselines:

- **FT**$_{Doc}$ tests a "continued training" scenario: models finetune on the raw documentation for the 100 most used Excel functions, which contain simple examples paired with toy tables. However, the documentation format used (including toy examples) does not mirror the format of the downstream QA task, which may be suboptimal for the student model.
- **FT**$_{Doc-QA}$ addresses format by using GPT-4o to reformat extracted examples from the documentation into the same QA format of the downstream task. This involves purely restructuring natural language content only, and no generation of novel, synthetic content.

We also use the **FT**$_{Doc-QA}$ baseline as a control for understanding our synthetic training data: because the format is the same, improvement over this baseline must be attributed to the *content* which we generate using our method.

5. Results and Discussion

5.1. WikiTQ

Table 1 shows full results on WikiTQ.

In RAG_{All} all models except for CodeLlama 13B suffer when given all function signatures, likely because the long context obscures the most relevant function. However, optimal retrievals in RAG_{Oracle} improve most models, showing the upper-bound of a RAG approach. However, improvement is inconsistent and not large over *Base Models*.

In the finetuned FT_{Doc} setting,"continued training" on raw function docs often harms performance. However, restructuring this into QA format (FT_{Doc-QA}) can sometimes be useful, such as in the weaker LLama2 models. Yet compared to *Base Models* the improvement is also marginal.

	Base Model	FT_{Syn-QA}
GPT-4o	77.78	-
Qwen2.5-coder 3B	5.88	**8.06**
Qwen2.5-coder 14B	14.37	**14.60**
Qwen2.5 3B	6.75	**11.11**
Qwen2.5 14B	**15.47**	15.03
CodeLlama-Instruct 7B	0.44	**2.18**
CodeLlama-Instruct 13B	3.49	**7.41**
Llama2 7B	0.00	**3.92**
Llama2 13B	1.09	**6.54**

Table 3: Evaluation results on our subset of TAT-QA. Execution Match (EM) results are shown.

In FT_{Syn-QA} we achieve a significant performance boost over all *Base Models* by finetuning on our synthetic data. The finetuned models even outperform corresponding RAG_{Oracle} settings, where the functions expected to be used are provided as hints by oracle o1. See Figure 1 for a showcase.

Regarding code pretraining, it is not clear that base models pretrained in HRPLs exhibit any consistent improvement in generating Excel, an LRPL. However, most code-pretrained models enjoy a greater boost from synthetic data finetuning compared to non-specialized counterparts. This shows clearly that previously learned coding skills in HRPLs can transfer to LRPLs.

5.2. TAT-QA

We show results of evaluating base models and finetuned models on TAT-QA in Table 3.

We observe similar trends on TAT-QA as we do on WikiTQ. Large models perform better than small models, and our finetuning data is able to improve 7 of 8 models significantly. We notice that in general, the effect size of finetuning is smaller when evaluated on TAT-QA, and we look to the lower base model performances to explain this. While TAT-QA is a similarly-formatted dataset in which data tables are paired with natural language queries, it is in a very specialized domain. Financial data is likely to be more scarce in general LLM pretraining data, so we expect a priori that performance may be lower on this dataset, and the lower base model performances compared to WikiTQ confirm this.

We also note that it is unclear if code-specialized models outperform non-code-specialized models on this task. We believe this is for the same reason as above: code-specialization does not appear to improve models' financial reasoning ability, and our further code-finetuning in the general domain of Wikipedia tables may not show particular benefits

Result Category	Base Model	FT_{Syn-QA}
Correct	13	34
Error: Plan Logic	25	33
Error: Function Choice	8	9
Error: Table Indexing	35	13
Error: Formula Syntax	9	1
Error: Other	10	10

Table 4: Analysis of errors by Qwen2.5-Coder 3B base and finetuned on 100 WikiTQ test samples.

Result Category	Guideline
Correct	The model generation executes to the correct answer
Error: Plan Logic	If the CoT is wrong
Error: Function Choice	If the CoT is correct, but it decides to use the wrong function, or is missing some functions
Error: Formula Syntax	If the function is correct, but the way of using the function is wrong
Error: Table Indexing	If the function usage is correct, and changing the cell/row/column numbers would get the correct answer
Error: Other	Any other error, such as correct plan but incorrect formula semantics

Table 5: Annotation guide for error classification.

to coding models on this dataset, either.

5.3. Error Analysis of Model Generations

We randomly sample 100 problems from the WikiTQ test set and conduct a qualitative analysis of the model generations produced by Qwen2.5-Coder 3B *Base Model* and FT_{Syn-QA}.[5] Table 4 demonstrates that fine-tuning on synthetic data primarily reduces errors in table reading (-22 questions) and formula syntax (-8), while preserving the model's ability in planning and function selection. The small increase in planning errors is attributed to the model's tendency to produce single-function formulas. Since the synthetic data focuses on the use of individual functions, we observe that 64.3% of the solutions generated by FT_{Syn-QA} involve a single function, compared to 44.4% in the base model. In our error analysis we use the categorization guidelines in Table 5 to analyze the generations from Qwen models on our WikiTQ test set.

Next, we turn to analyzing model performance before and after our function finetuning. We first show the percentage of model-generated formulas which consist of a single function call in Table 6. We observe that all models increased the number of single function predictions after finetuning, however this effect is more noticeable in smaller models.

[5]This code-pretrained model was chosen for its large performance boost in testing.

	Base Model	FT_{Syn-QA}
Qwen2.5-coder 3B	44.44	64.32
Qwen2.5-coder 14B	46.17	59.00
Qwen2.5 3B	40.22	73.08
Qwen2.5 14B	40.22	40.85

Table 6: Percentage of generated formulas consisting of a single function.

	Improved	Regressed
Qwen2.5-coder 3B	29.77	10.87
Qwen2.5-coder 14B	27.27	7.14
Qwen2.5 3B	23.66	3.64
Qwen2.5 14B	17.33	21.92

Table 7: Improvements and regressions when the student model predicts the same, single-function formula before and after finetuning. Models master individual functions and often improve in generating single-function calls, usually with little regression.

Shown in Table 7, of each model's improvements (samples which were incorrect before tuning, and correct after), many generations produce the same function before and after. This shows learned mastery of these functions, where finetuned models produce better formulas using the same function. We also observe minimal regressions when predicting the same function (correct usage before tuning, but incorrect after).

6. Conclusion

We show that generating textbook-quality examples for a low resource programming language can rapidly improve LLM generation through finetuning, with benefits over RAG approaches and other, less rich finetuning paradigms. Yet our method is simple and automatic, requiring only function documentation, a teacher model generally strong in instruction-following (which need not be an expert in the target domain), and minimal human description of the domain. We demonstrate the method for Excel Formula language which is popular in practice, but an LRPL owing to its sparsity of training data. Our method improves student models by leveraging their problem solving abilities previously learned in pretraining on high-resource programming languages to adapt to the new domain.

We posit that further work on individual languages may benefit similarly, and we encourage future work in synthetic data generation for LLM pretraining, especially for "long-tail" phenomena like LRPLs which suffer due to a lack of natural training data.

Limitations

We demonstrated our methods on a single LRPL, though our findings should apply to any low-resource programming language due to the generality of our method. We also focused on generating demonstrations for just the function library in our example LRPL, a more realistic target for generating textbook-quality data than the infinite space of possible programs, especially when limited resources exist in the target domain. While we cannot compare to the challenges or benefits of generating complex data, as we noted in error analysis this limitation clearly biases models towards simpler solutions, so this presents a problem which future research may alleviate. Further, due to computational costs we only investigated methods for pruning bad synthetic generations, and not any methods for fixing those bad generations. Research in this direction could also prove valuable.

7. Bibliographical References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 technical report: A highly capable language model locally on your phone.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha,

Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Marc Fisher and Gregg Rothermel. 2005. The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks are all you need.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2023. Unnatural instructions: Tuning language models with (almost) no human labor. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14409–14428, Toronto, Canada. Association for Computational Linguistics.

Sathvik Joel, Jie JW Wu, and Fatemeh H. Fard. 2024. A survey on llm-based code generation for low-resource and domain-specific programming languages.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The stack: 3 tb of permissively licensed source code. *Preprint*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. Starcoder 2 and the stack v2: The next generation.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Alexandre Magueresse, Vincent Carles, and Evan Heetderks. 2020. Low-resource languages: A review of past work and future challenges.

Microsoft Support. 2024. Excel functions. https://support.microsoft.com/en-gb/office/excel-functions-alphabetical-b3944572-255d-4efb-bb96-c6d90033e188. [Accessed 17-05-2025].

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely,

David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. 2024a. Openai o1 system card.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKin-

ney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024b. Gpt-4 technical report.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables.

Justin Payan, Swaroop Mishra, Mukul Singh, Carina Negreanu, Christian Poelitz, Chitta Baral, Subhro Roy, Rasika Chakravarthy, Benjamin Van Durme, and Elnaz Nouri. 2023. InstructExcel: A benchmark for natural language instruction in excel. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4026–4043, Singapore. Association for Computational Linguistics.

Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in Text+Code LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11234–11258, Miami, Florida, USA. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report.

Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woomyoung Park. 2021. GPT3Mix: Leveraging large-scale language models for text augmentation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2225–2239, Punta Cana,

Dominican Republic. Association for Computational Linguistics.

Yueheng Zhang, Xiaoyuan Liu, Yiyou Sun, Atheer Alharbi, Hend Alzahrani, Basel Alomair, and Dawn Song. 2025. Can llms design good questions based on context?

Wei Zhao, Zhitao Hou, Siyuan Wu, Yan Gao, Haoyu Dong, Yao Wan, Hongyu Zhang, Yulei Sui, and Haidong Zhang. 2024. NL2Formula: Generating spreadsheet formulas from natural language queries. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2377–2388, St. Julian's, Malta. Association for Computational Linguistics.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.