Contents

1 Experiments Results

1.1 Original in Thesis

Stages	Token consumption (tks)				Time consumption (s)				Success rate
	Average	Minimum	Maximum	Median	Average	Minimum	Maximum	Median	
Data processing	16500.56	13072	21032	14983.50	34.92	24.47	48.33	33.28	0.8
Model INT8 Quantization & conversion	5318.17	613	14146	3597.50	21.39	7.27	44.32	19.00	0.9
Sketch generation	31184.80	22448	37076	33518.00	109.21	93.17	143.20	98.43	0.25

Table 1: Token and time consumption count in different stages.

1.2 Results: 2024.08.09, Model: GPT-4o

This test was conducted through 30 runs of each task, and the statistics include both passed and failed runs.

Stages	Total Token Consumption				Input Token Consumption				Output Token Consumption			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
Data Processing	16621	15616	38380	817	14740	13930	32402	612	1880	1652	5978	205
Model INT8 Quantization & conversion	4700	3698	17020	582	3916	3043	14927	333	784	662	2434	249
Sketch Generation	29858	30542	46780	12759	25675	26457	42591	9990	4182	4245	5931	2496

Table 2: Token consumption.

Stages	Total Time (s)			
	Ave	Med	Max	Min
Data Processing	39.38	30.39	121.11	2.81
Model INT8 Quantization & conversion	18.62	14.78	48.33	6.34
Sketch Generation	100.99	108.82	147.56	64.41

Table 3: Time consumption.

Stages	Total Cost ($)				Input Cost ($)				Output Cost ($)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
Data Processing	0.0557	0.0517	0.1408	0.0036	0.0369	0.0348	0.0810	0.0015	0.0188	0.0165	0.0598	0.0021
Model INT8 Quantization & conversion	0.0176	0.0142	0.0588	0.0033	0.0098	0.0076	0.0373	0.0008	0.0078	0.0066	0.0243	0.0025
Sketch Generation	0.1078	0.1100	0.1484	0.0527	0.0653	0.0682	0.1065	0.0250	0.0424	0.0429	0.0593	0.0250

Table 4: Cost in US dollars.

Stages	Ave. Time (s)		Ave. Total Tokens		Success Rate
	Passed	Failed	Passed	Failed	
Data Processing	36.00	69.79	15284.30	28654.00	0.900
Model INT8 Quantization & conversion	16.63	46.58	3849.43	16622.00	0.933
Sketch Generation	113.86	119.58	10094.89	14286.14	0.300

Table 5: Performance metrics regarding passed and failed runs.

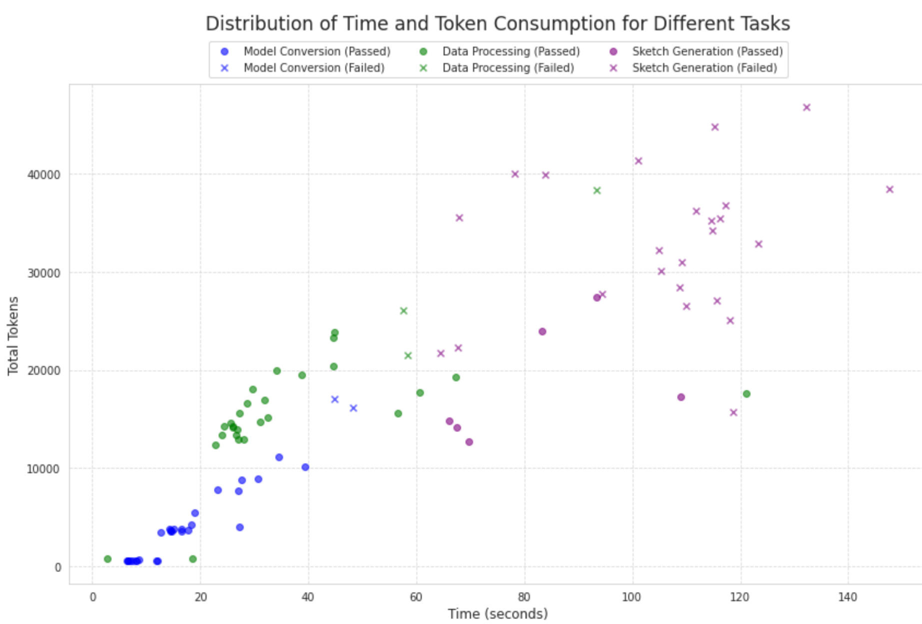


Figure 1: Distribution of time and token consumption in three tasks.

1.3 Results: 2024.10.02, Model: GPT-4o

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	16621	15616	38380	817	14740	13930	32402	612	1880	1652	5978	205
MC	4700	3698	17020	582	3916	3043	14927	333	784	662	2434	249
SG	13028	13747	16602	1731	9382	9829	12652	1255	3646	3932	4874	476

Table 6: Token consumption details regarding Overall, Input, and Output in three stages. 10.02

Stages	Execution Time (s)			Token Consumption			Success
	Ave	Max	Min	Ave	Max	Min	Rate (%)
DP	39.38	121.11	2.81	16621	38380	817	0.900
MC	18.62	48.33	6.34	4700	17020	582	0.933
SG	117.86	242.87	7.13	13028	16602	1731	0.300

Table 7: Performance metrics including execution time, token consumption, and success rate in three stages. 10.02

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	5.57	5.17	14.08	0.36	3.69	3.48	8.10	0.15	1.88	1.65	5.98	0.21
MC	29.39	23.67	98.03	5.54	16.32	12.68	62.20	1.39	13.07	11.03	40.57	4.15
SG	10.16	10.84	12.58	1.34	4.69	4.91	6.33	0.63	5.47	5.90	7.31	0.71

Table 8: Cost details regarding Overall, Input, and Output in three stages. 10.02

Stages	Ave Execution Time (s)			Ave Total Tokens			Success
	Overall	Pass	Fail	Overall	Pass	Fail	Rate (%)
DP	39.38	36.00	69.79	16621	15284	28654	0.900
MC	18.62	16.63	46.58	4700	3849	16622	0.933
SG	117.86	113.86	119.58	13028	10094	14286	0.300

Table 9: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. 10.02

Stages	Ave	Med	Max	Min
DP	39.38	30.39	121.11	2.81
MC	18.62	14.78	48.33	6.34
SG	117.86	115.15	242.87	7.13

Table 10: Time consumption (seconds) in three stages. 10.02

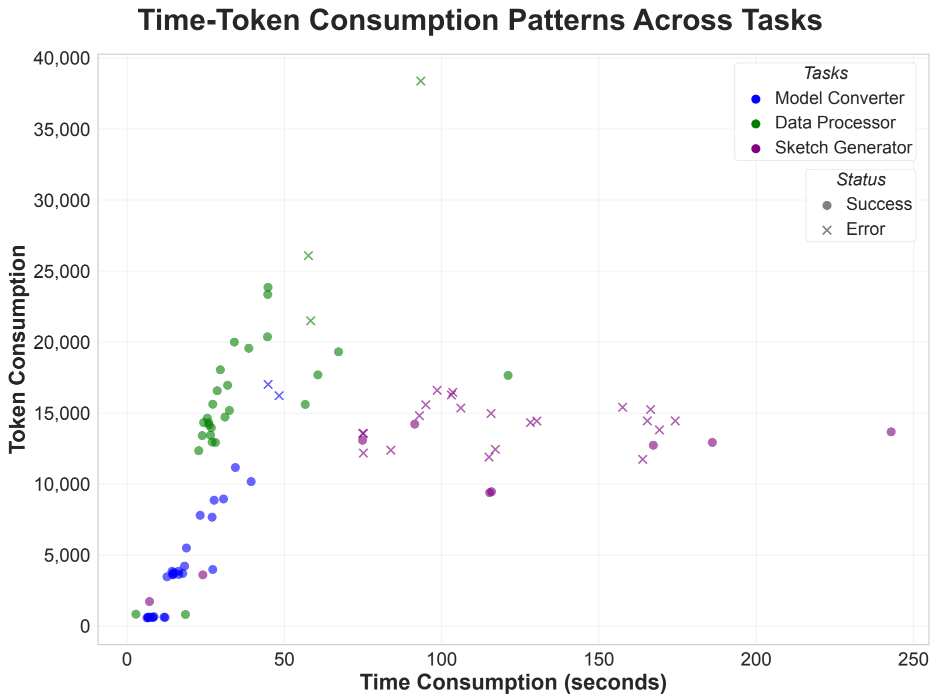


Figure 2: Distribution of time and token consumption in three tasks 10.02.

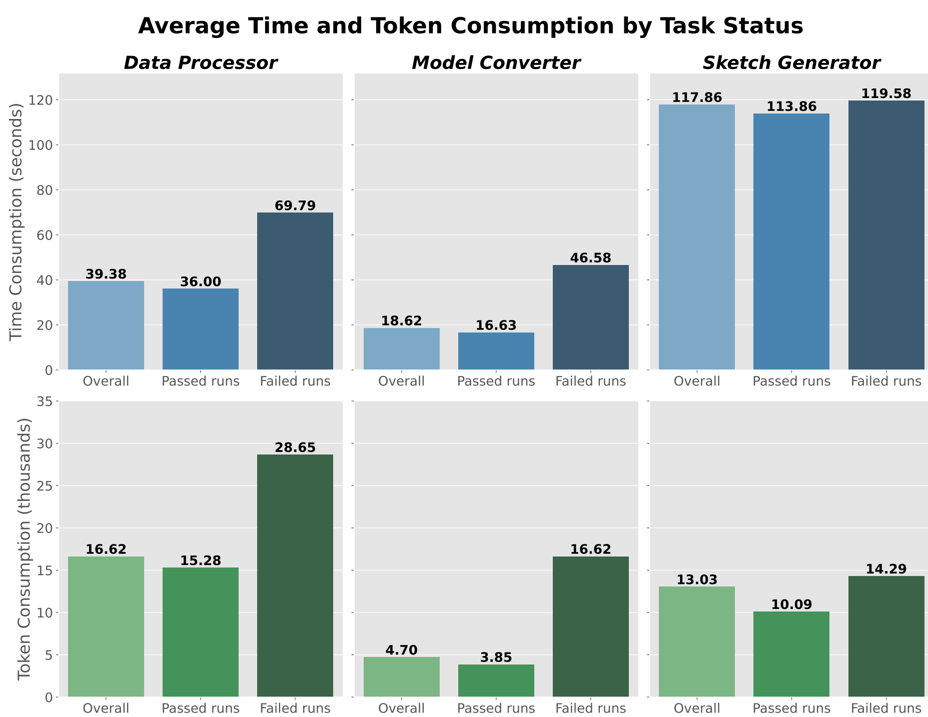


Figure 3: Average Time and Token Consumption by Task Status.
The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs. 10.02.

1.4 Results: 2024.10.25, Model: GPT-4o

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	11074	10446	21777	8587	8515	8076	17517	6688	2559	2369	4447	1831
MC	793	567	4035	534	508	299	3463	299	284	268	572	235
SG	13028	13747	16602	1731	9382	9829	12652	1255	3646	3932	4874	476

Table 11: Token consumption details regarding Overall, Input, and Output in three stages. 10.25

Stages	Execution Time (s)			Token Consumption			Success
	Ave	Max	Min	Ave	Max	Min	Rate (%)
DP	44.17	73.88	31.68	11074	21777	8587	1.000
MC	8.87	22.08	6.66	793	4035	534	1.000
SG	117.86	242.87	7.13	13028	16602	1731	0.300

Table 12: Performance metrics including execution time, token consumption, and success rate in three stages. 10.25

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	8.10	7.54	15.15	6.12	4.26	4.04	8.76	3.34	3.84	3.55	6.67	2.75
MC	0.68	0.55	2.59	0.50	0.25	0.15	1.73	0.15	0.43	0.40	0.86	0.35
SG	10.16	10.84	12.58	1.34	4.69	4.91	6.33	0.63	5.47	5.90	7.31	0.71

Table 13: Cost details regarding Overall, Input, and Output in three stages. 10.25

Stages	Ave Execution Time (s)			Ave Total Tokens			Success
	Overall	Pass	Fail	Overall	Pass	Fail	Rate (%)
DP	44.17	44.17	0.00	11074	11074	0	1.000
MC	8.87	8.87	0.00	793	793	0	1.000
SG	117.86	113.86	119.58	13028	10094	14286	0.300

Table 14: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. 10.25

Stages	Ave	Med	Max	Min
DP	44.17	40.21	73.88	31.68
MC	8.87	7.98	22.08	6.66
SG	117.86	115.15	242.87	7.13

Table 15: Time consumption (seconds) in three stages. 10.25

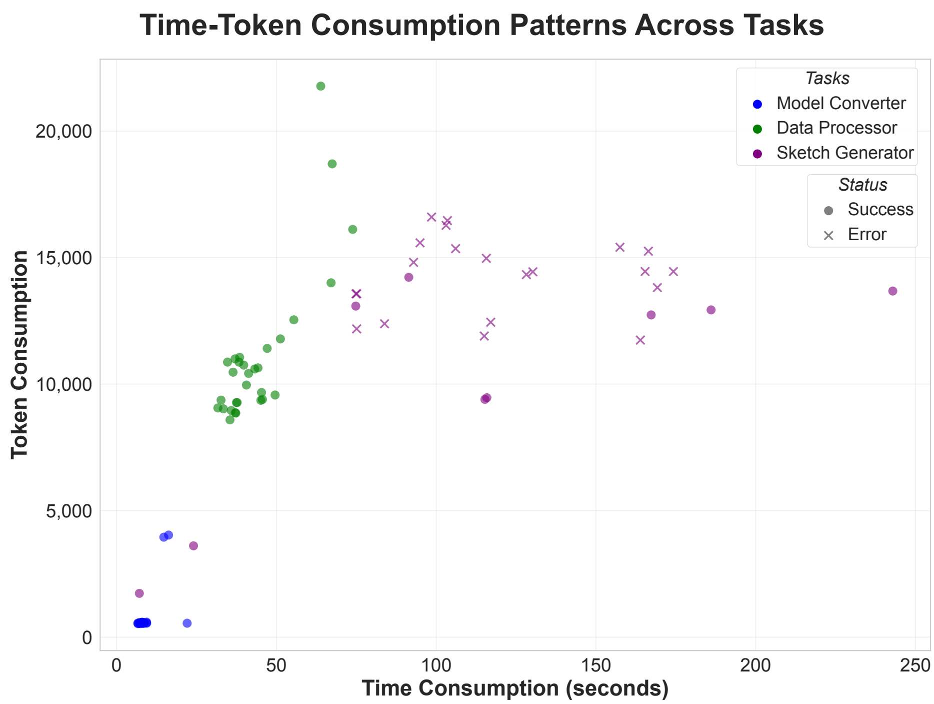


Figure 4: Distribution of time and token consumption in three tasks 10.25.

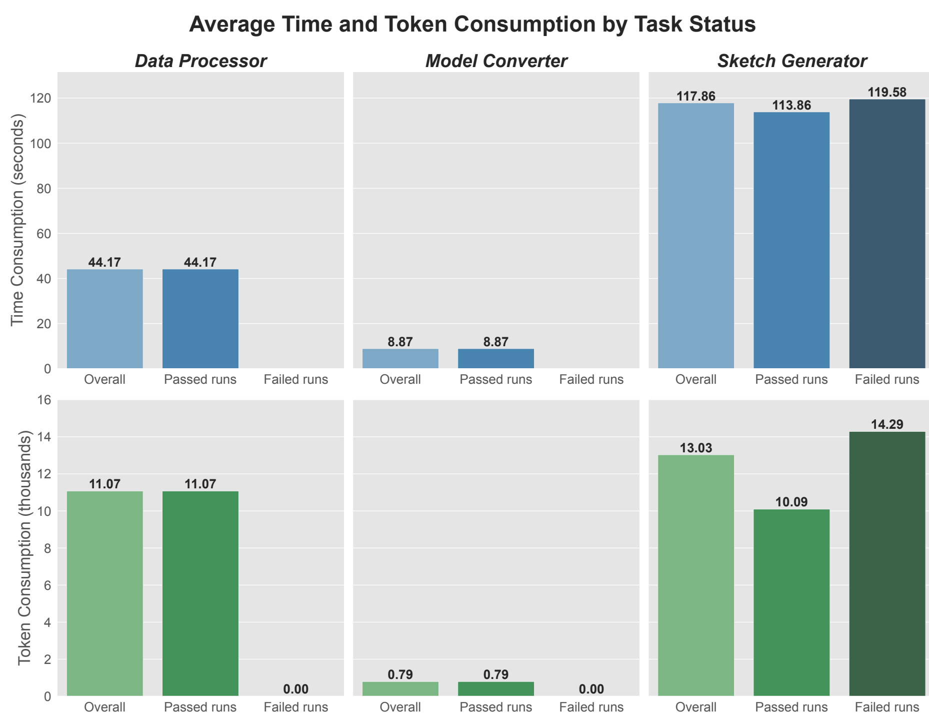


Figure 5: Average Time and Token Consumption by Task Status.
The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs. 10.25.

1.5 Results: 2024.10.28, Model: GPT-4o

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	10832	9433	25086	8560	8273	7030	18952	6562	2558	2319	6134	1953
MC	688	573	3949	545	403	299	3433	299	285	274	516	246
SG	13320	13906	17181	1840	9606	10011	13021	1361	3714	3916	4645	479

Table 16: Token consumption details regarding Overall, Input, and Output in three stages. 10.28

Stages	Execution Time (s)			Token Consumption			Success
	Ave	Max	Min	Ave	Max	Min	Rate (%)
DP	47.76	155.93	32.58	10832	25086	8560	0.900
MC	6.09	10.21	3.65	688	3949	545	1.000
SG	60.55	87.92	7.73	13320	17181	1840	0.367

Table 17: Performance metrics including execution time, token consumption, and success rate in three stages. 10.28

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	7.97	7.17	18.68	6.28	4.14	3.52	9.48	3.28	3.84	3.48	9.20	2.93
MC	0.63	0.56	2.49	0.52	0.20	0.15	1.72	0.15	0.43	0.41	0.77	0.37
SG	10.38	11.03	12.75	1.40	4.80	5.01	6.51	0.68	5.57	5.87	6.97	0.72

Table 18: Cost details regarding Overall, Input, and Output in three stages. 10.28

Stages	Ave Execution Time (s)			Ave Total Tokens			Success
	Overall	Pass	Fail	Overall	Pass	Fail	Rate (%)
DP	47.76	44.75	74.82	10832	9731	20744	0.900
MC	6.09	6.09	0.00	688	688	0	1.000
SG	60.55	48.75	67.38	13320	11131	14588	0.367

Table 19: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. 10.28

Stages	Ave	Med	Max	Min
DP	47.76	39.69	155.93	32.58
MC	6.09	5.92	10.21	3.65
SG	60.55	64.74	87.92	7.73

Table 20: Time consumption (seconds) in three stages. 10.28

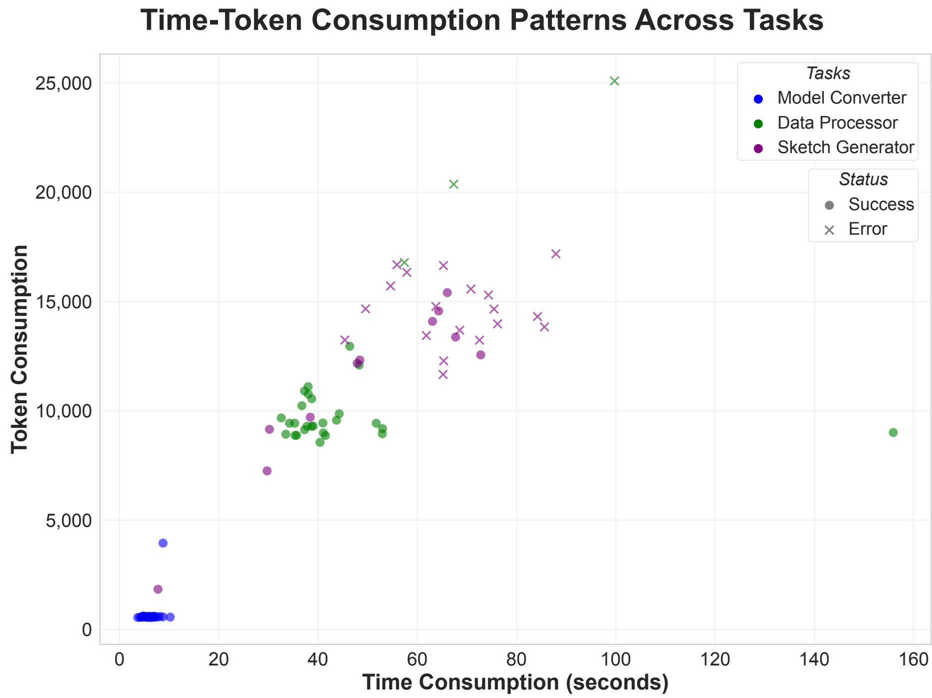


Figure 6: Distribution of time and token consumption in three tasks 10.28.

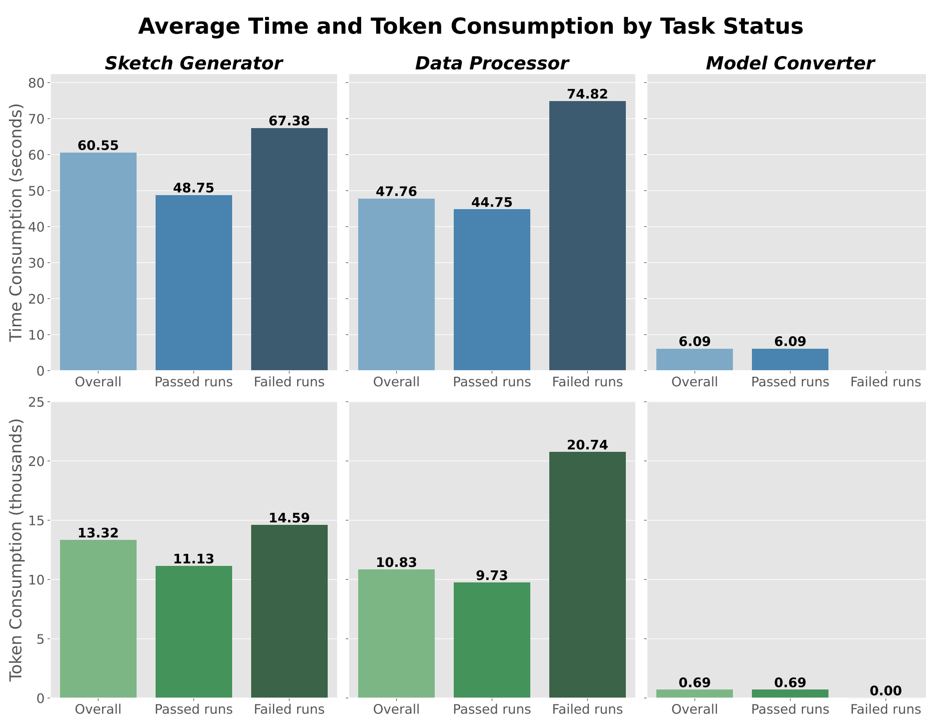


Figure 7: Average Time and Token Consumption by Task Status.
The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs. 10.28.

1.6 Results: 2024.11.14, Model: GPT-4o: Comparison of the impact of JSON structure on SG

NOTICE: The results were based on the task of sketch generation with 3 levels of JSON structure usage. From 0 to 2, higher number indicates fewer JSON structures.

1 Sketch Generation0 or SG0 is the original prompt structure, with the `guideline` embedded in the `application_specification`, which again embedded in the task description (`task`), under ### OBJECTIVE ###. They three are all JSON-like objects.

```
### OBJECTIVE ###
"task":{
    ...
    "application_specification":{
        ...
        "guideline":{
            ...
        }
    }
}
```

2 Sketch Generation1 or SG1 is the prompt structure with the task description directly written as a prompt section (noted as '### OBJECTIVE ###') in Markdown format, and the `application_specification` written as another prompt section (noted as `### APPLICATION SPECIFICATIONS ###`) but still as a JSON-like object. The `guideline`, as a JSON-like object, embedded in the `application_specification`, remained the same.

```
### OBJECTIVE ###
- **TASK**:...
...

### APPLICATION SPECIFICATIONS ###
"application_specification":{
        ...
        "guideline":{
            ...
        }
    }
```

3 Sketch Generation2 or SG2 is the prompt structure similar to SG1, only the `guideline` was moved out as a prompt section (noted as `### GUIDELINE FOR SKETCH ###`) in Markdown format.

```
### OBJECTIVE ###
- **TASK**:...
...

### APPLICATION SPECIFICATIONS ###
"application_specification":{
        ...
}

### GUIDELINE FOR SKETCH ###
# Steps...
...
```

As a result:

Average Token Comsumption wise: $SG0 \gg SG2 > SG1$

Success Rate wise: $SG1 > SG0 \gg SG2$

Average Time Consumption wise: $SG2 > SG1 > SG0$

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
SG0	14435	15709	16953	9991	10262	10897	12641	6928	4172	4192	5095	2914
SG1	18014	18298	23105	4739	14243	14344	18631	4181	3770	4039	4497	558
SG2	18671	19890	26742	7852	14707	15515	22237	5988	3964	4157	4657	1864

Table 21: Token consumption details regarding Overall, Input, and Output in three levels of JSON structure usage. 11.14

Stages	Execution Time (s)			Token Consumption			Success
	Ave	Max	Min	Ave	Max	Min	Rate (%)
SG0	87.42	174.55	44.50	14435	16953	9991	0.250
SG1	42.76	76.44	7.03	18014	23105	4739	0.300
SG2	53.39	145.49	22.41	18671	26742	7852	0.150

Table 22: Performance metrics including execution time, token consumption, and success rate in three levels of JSON structure usage. 11.14

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
SG0	6.74	7.06	8.00	4.68	2.57	2.72	3.16	1.73	4.17	4.19	5.10	2.91
SG1	7.33	7.54	9.13	1.60	3.56	3.59	4.66	1.05	3.77	4.04	4.50	0.56
SG2	7.64	8.11	10.06	3.36	3.68	3.88	5.56	1.50	3.96	4.16	4.66	1.86

Table 23: Cost details regarding Overall, Input, and Output in three levels of JSON structure usage. 11.14

Stages	Ave Execution Time (s)			Ave Total Tokens			Success
	Overall	Pass	Fail	Overall	Pass	Fail	Rate (%)
SG0	87.42	85.02	88.22	14435	12356	15128	0.250
SG1	42.76	38.18	44.72	18014	17299	18320	0.300
SG2	53.39	30.87	57.36	18671	11522	19933	0.150

Table 24: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three levels of JSON structure usage. 11.14

Stages	Ave	Med	Max	Min
SG0	87.42	73.25	174.55	44.50
SG1	42.76	43.31	76.44	7.03
SG2	53.39	46.85	145.49	22.41

Table 25: Time consumption (seconds) in three levels of JSON structure usage. 11.14

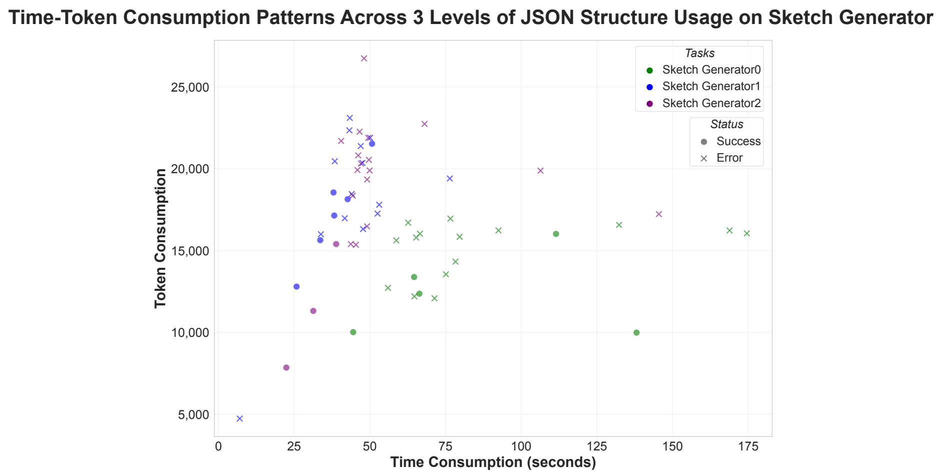


Figure 8: Distribution of time and token consumption in three levels of JSON structure. Model: GPT-4o, Date: 11.14.

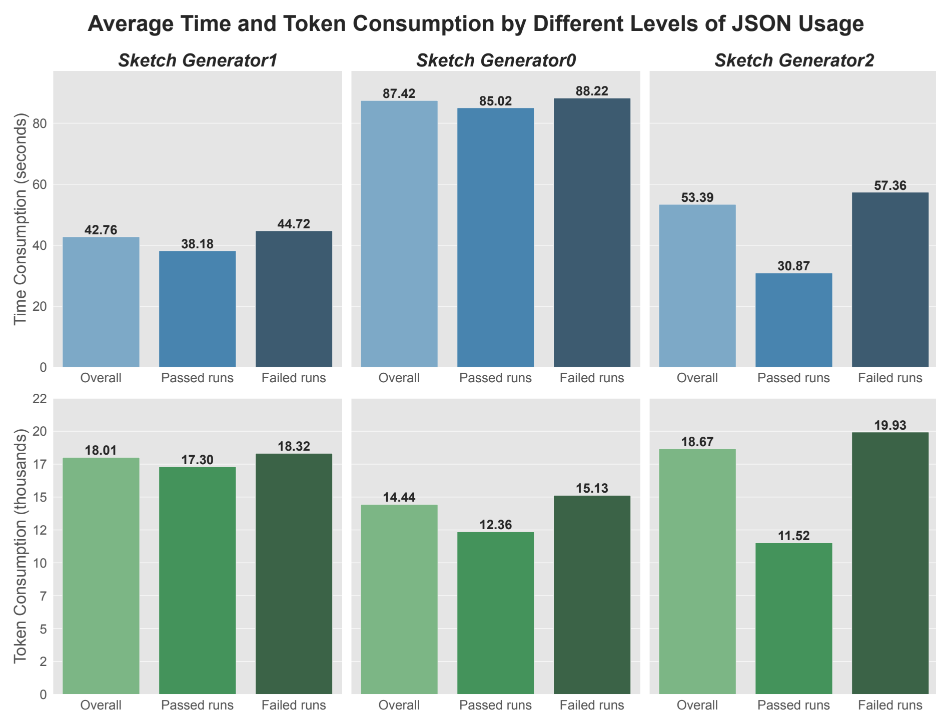


Figure 9: Average Time and Token Consumption by different levels of JSON Usage. Model: GPT-4o, Date: 11.14.

The average time (top) and token consumption (bottom) are compared across three levels of JSON structure, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs.

1.7 Results: 2024.11.28, Model: GPT-4o-mini

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	21693	19154	36578	17390	16611	14323	28000	13604	5081	4759	9102	3710
MC	5843	3949	12801	519	5014	3439	11521	299	828	510	1691	220
SG	27406	30304	33098	5172	21539	23980	25266	3934	5866	6351	7832	1238

Table 26: Token consumption details regarding Overall, Input, and Output in three stages. Model: GPT-4o-mini, Date: 11.28

Stages	Execution Time (s)			Token Consumption			Success
	Ave	Max	Min	Ave	Max	Min	Rate (%)
DP	91.24	149.04	60.04	21693	36578	17390	0.900
MC	12.57	34.66	3.02	5843	12801	519	0.750
SG	73.93	114.93	15.25	27406	33098	5172	0.100

Table 27: Performance metrics including execution time, token consumption, and success rate in three stages. Model: GPT-4o-mini, Date: 11.28

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	0.55	0.50	0.96	0.43	0.25	0.21	0.42	0.20	0.30	0.29	0.55	0.22
MC	0.12	0.08	0.25	0.02	0.08	0.05	0.17	0.00	0.05	0.03	0.10	0.01
SG	0.68	0.74	0.85	0.13	0.32	0.36	0.38	0.06	0.35	0.38	0.47	0.07

Table 28: Cost details regarding Overall, Input, and Output in three stages. Model: GPT-4o-mini, Date: 11.28

Stages	Ave Execution Time (s)			Ave Total Tokens			Success
	Overall	Pass	Fail	Overall	Pass	Fail	Rate (%)
DP	91.24	91.03	93.17	21693	21023	27723	0.900
MC	12.57	9.08	23.03	5843	4413	10131	0.750
SG	73.93	15.75	80.39	27406	5213	29872	0.100

Table 29: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. Model: GPT-4o-mini, Date: 11.28

Stages	Ave	Med	Max	Min
DP	91.24	85.82	149.04	60.04
MC	12.57	8.89	34.66	3.02
SG	73.93	76.27	114.93	15.25

Table 30: Time consumption (seconds) in three stages. Model: GPT-4o-mini, Date: 11.28

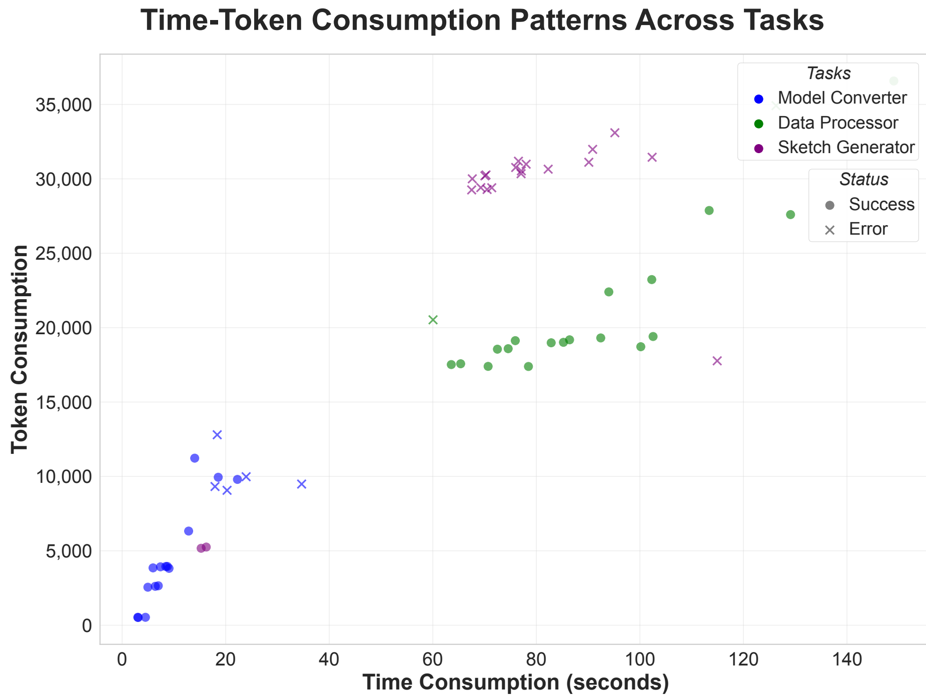


Figure 10: Distribution of time and token consumption in three tasks. Model: GPT-4o-mini, Date: 11.28.

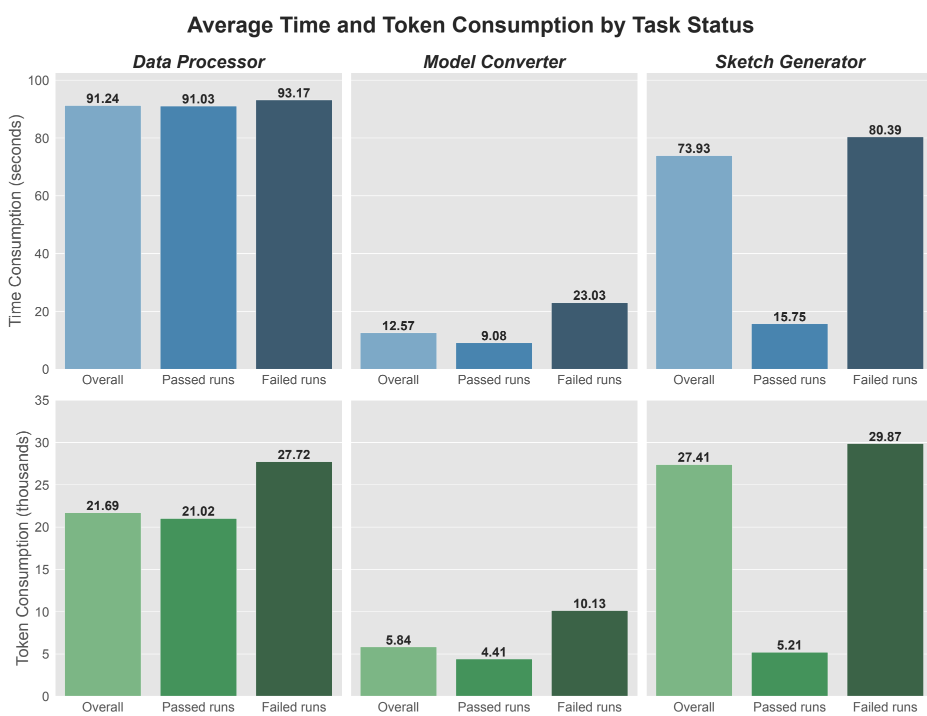


Figure 11: Average Time and Token Consumption by Task Status. Model: GPT-4o-mini, Date: 11.28. *The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs.*

1.8 Results: 2025.01.15, Model: phi4:14b

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	15220	14719	28556	11309	12031	11350	22110	8967	3189	2928	6446	2131
MC	2149	767	8731	674	1463	389	6561	389	686	378	2170	285

Table 31: Token consumption details regarding Overall, Input, and Output in three stages. Model: phi4:14b, Date: 01.15

Stages	Execution Time (s)			Token Consumption			Success Rate
	Ave	Max	Min	Ave	Max	Min	
DP	57.70	116.58	38.09	15220	28556	11309	0.500
MC	20.09	57.85	8.52	2149	8731	674	0.967

Table 32: Performance metrics including execution time, token consumption, and success rate in three stages. Model: phi4:14b, Date: 01.15

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 33: Cost details regarding Overall, Input, and Output in three stages. Model: phi4:14b, Date: 01.15

Stages	Ave Execution Time (s)			Ave Total Tokens			Success Rate
	Overall	Pass	Fail	Overall	Pass	Fail	
DP	57.70	47.11	68.29	15220	12867	17573	0.500
MC	20.09	19.03	50.70	2149	1922	8731	0.967

Table 34: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. Model: phi4:14b, Date: 01.15

Stages	Ave	Med	Max	Min
DP	57.70	50.04	116.58	38.09
MC	20.09	10.34	57.85	8.52

Table 35: Time consumption (seconds) in three stages. Model: phi4:14b, Date: 01.15

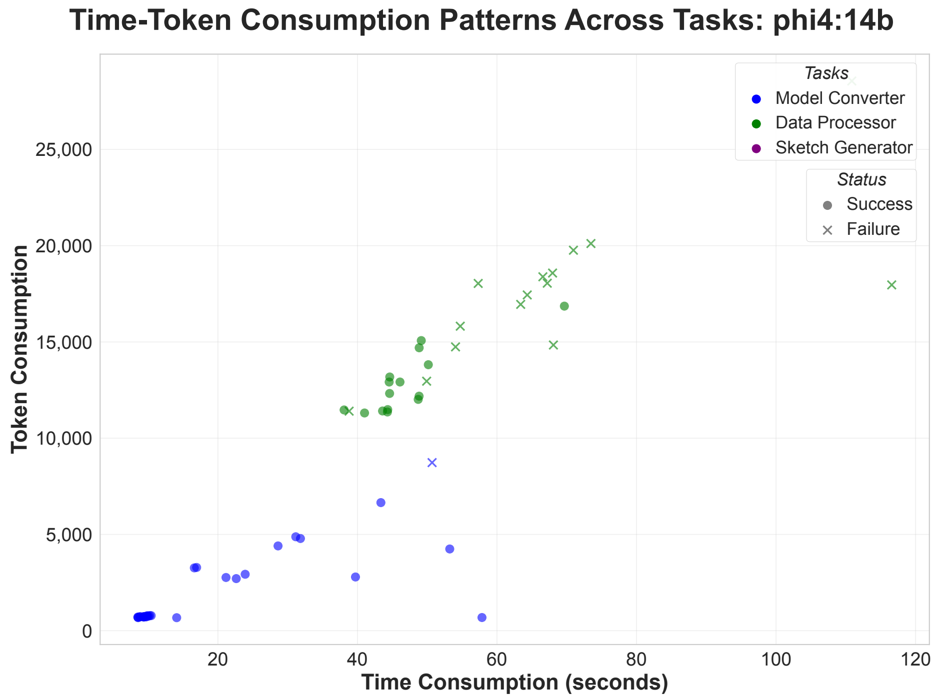


Figure 12: Distribution of time and token consumption in three tasks. Model: phi4:14b, Date: 01.15.

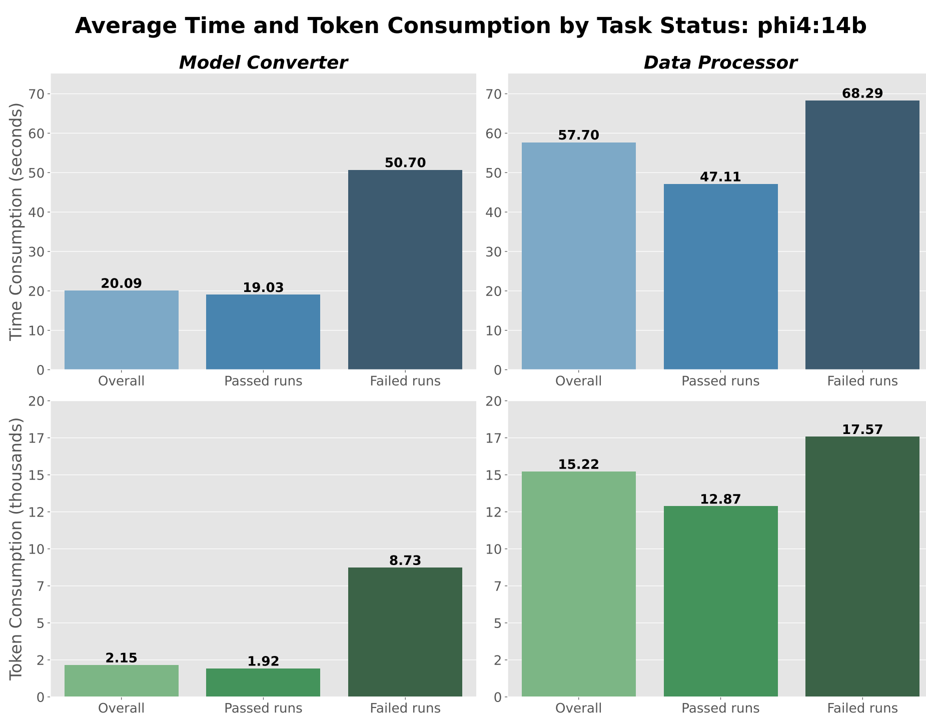


Figure 13: Average Time and Token Consumption by Task Status. Model: phi4:14b, Date: 01.15.
The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs.

1.9 Results: 2025.01.15, Model: llama3.1:8b

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	16869	16563	35068	2076	13469	13143	26029	1782	3400	3420	9039	294
MC	4636	5805	8555	523	3797	4880	6960	389	838	908	1595	134

Table 36: Token consumption details regarding Overall, Input, and Output in three stages. Model: llama3.1:8b, Date: 01.15

Stages	Execution Time (s)			Token Consumption			Success Rate
	Ave	Max	Min	Ave	Max	Min	
DP	47.73	123.03	3.55	16869	35068	2076	0.200
MC	25.42	41.55	6.54	4636	8555	523	0.633

Table 37: Performance metrics including execution time, token consumption, and success rate in three stages. Model: llama3.1:8b, Date: 01.15

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 38: Cost details regarding Overall, Input, and Output in three stages. Model: llama3.1:8b, Date: 01.15

Stages	Ave Execution Time (s)			Ave Total Tokens			Success Rate
	Overall	Pass	Fail	Overall	Pass	Fail	
DP	47.73	56.25	45.60	16869	17621	16681	0.200
MC	25.42	18.50	37.38	4636	3119	7256	0.633

Table 39: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. Model: llama3.1:8b, Date: 01.15

Stages	Ave	Med	Max	Min
DP	47.73	44.77	123.03	3.55
MC	25.42	31.89	41.55	6.54

Table 40: Time consumption (seconds) in three stages. Model: llama3.1:8b, Date: 01.15

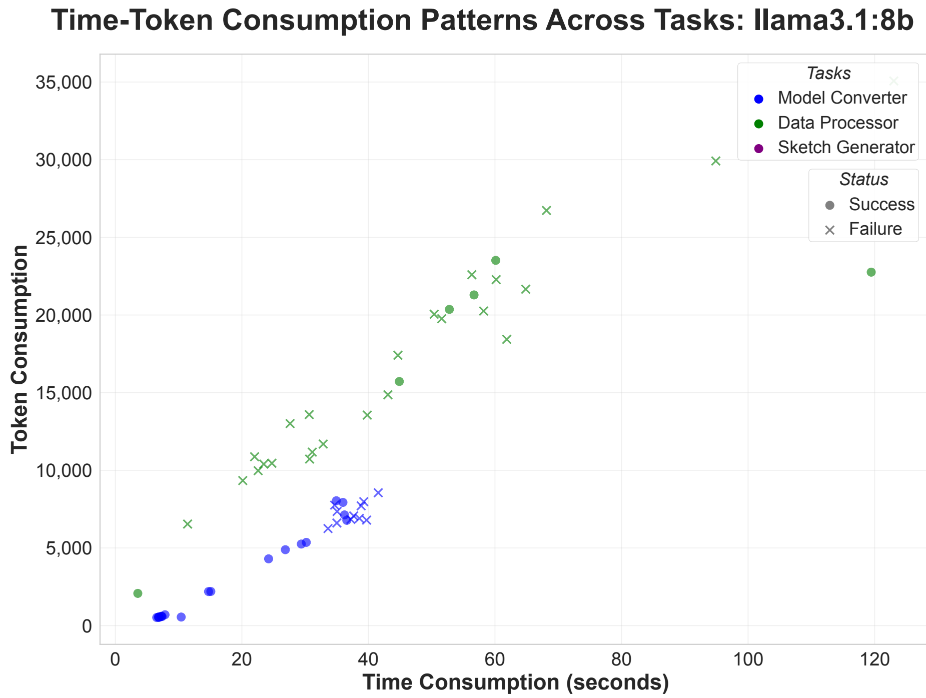


Figure 14: Distribution of time and token consumption in three tasks. Model: llama3.1:8b, Date: 01.15.

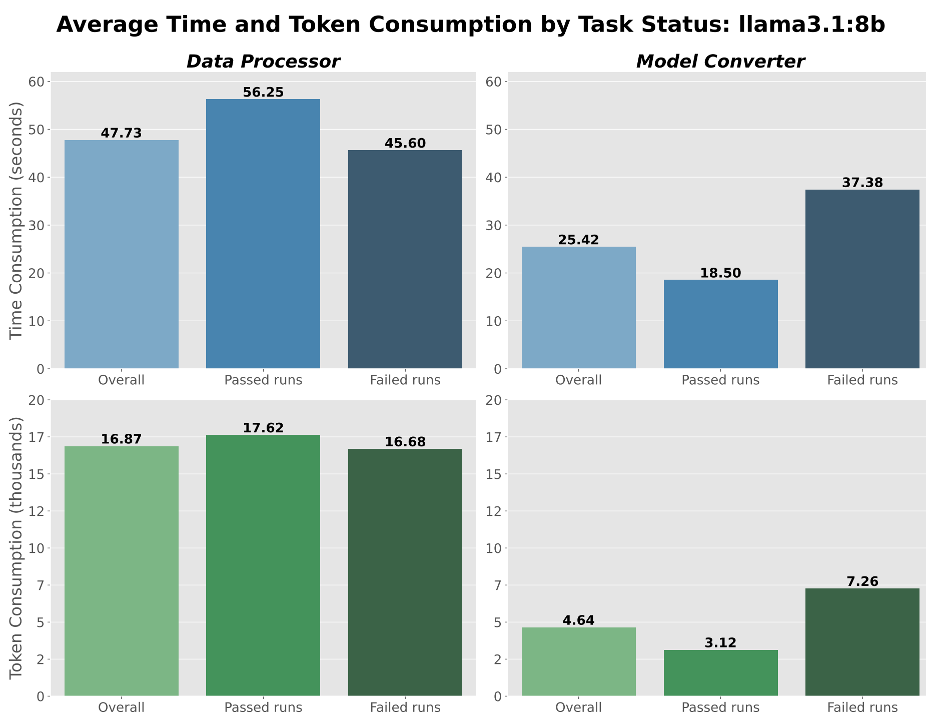


Figure 15: Average Time and Token Consumption by Task Status. Model: llama3.1:8b, Date: 01.15.
The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs.

1.10 Results: 2025.01.23, Model: qwen2.5-coder:14b

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	16929	16088	27965	12076	13991	13478	22244	10068	2937	2652	5745	1434
MC	6106	7162	9488	2420	4848	5640	7545	1942	1258	1505	2058	391

Table 41: Token consumption details regarding Overall, Input, and Output in three stages. Model: qwen2.5-coder:14b, Date: 01.23

Stages	Execution Time (s)			Token Consumption			Success Rate
	Ave	Max	Min	Ave	Max	Min	
DP	61.57	154.02	29.08	16929	27965	12076	0.300
MC	55.17	478.53	13.40	6106	9488	2420	0.567

Table 42: Performance metrics including execution time, token consumption, and success rate in three stages. Model: qwen2.5-coder:14b, Date: 01.23

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 43: Cost details regarding Overall, Input, and Output in three stages. Model: qwen2.5-coder:14b, Date: 01.23

Stages	Ave Execution Time (s)			Ave Total Tokens			Success Rate
	Overall	Pass	Fail	Overall	Pass	Fail	
DP	61.57	61.21	61.73	16929	14582	17935	0.300
MC	55.17	55.29	55.02	6106	4462	8257	0.567

Table 44: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. Model: qwen2.5-coder:14b, Date: 01.23

Stages	Ave	Med	Max	Min
DP	61.57	55.33	154.02	29.08
MC	55.17	41.22	478.53	13.40

Table 45: Time consumption (seconds) in three stages. Model: qwen2.5-coder:14b, Date: 01.23

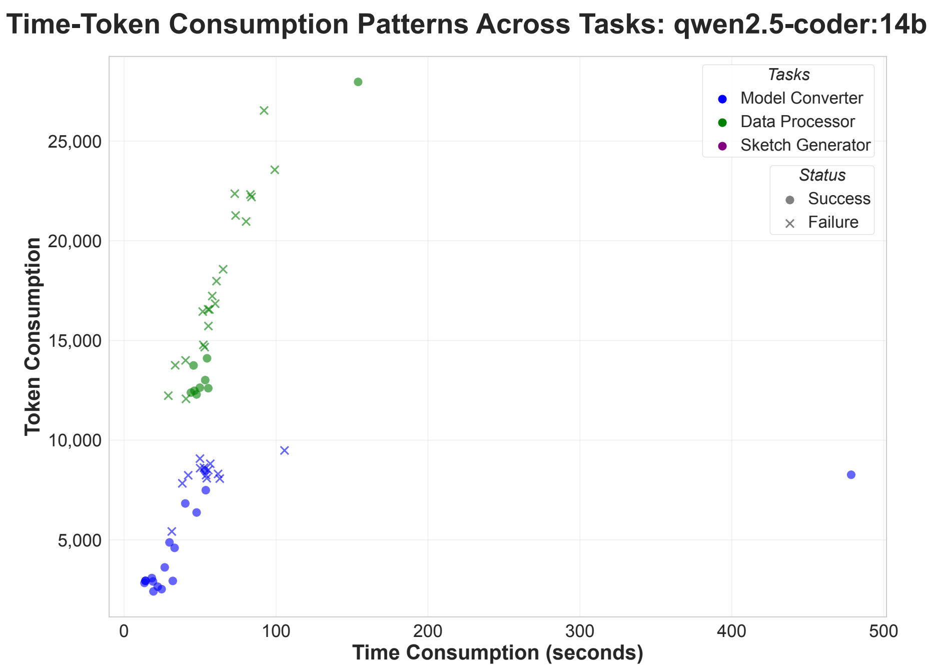


Figure 16: Distribution of time and token consumption in three tasks. Model: qwen2.5-coder:14b, Date: 01.23.

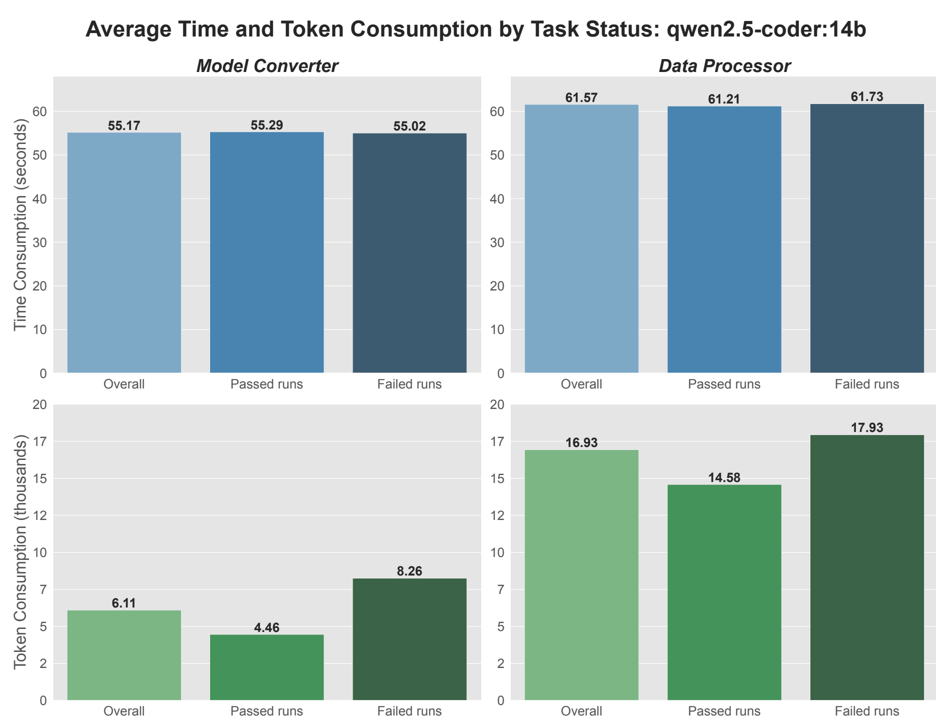


Figure 17: Average Time and Token Consumption by Task Status. Model: qwen2.5-coder:14b, Date: 01.23. *The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs.*

1.11 Results: 2025.01.23, Model: deepseek-r1:14b

In this test, we used the Deepseek's `deepseek-r1:14b`, which is a reasoning LLM in their reasoning series of LLMs. Similar to OpenAI's `o1` series, it trades off speed for more powerful reasoning ability. It also outputs the resoning process before the real answer to the question. A typical output is shown below 18. Although, the model is powerful at its resoning cabability according to recent studies and tests, our own test results that are based on using it's direct output as engineering code are not outstadning. Perhaps the resoning model series can be excellent for providing helpful insights in human-in-the-loop engineering tasks instead.

````
\<think\>
Alright, I need to help the user convert their Keras model to a TFLite model with 8-bit integer quantization. Let me break
down what needs to be done step by step.

First, the user provided specific paths for the original and converted models. So I'll start by importing the necessary
libraries: tensorflow as tf, and os for file handling.

Next, I need to load the original Keras model using tf.keras.load_model(). This should be straightforward as long as the
model is saved correctly.

Now, the main task is quantization. Since the user wants 8-bit integer quantization, I'll use the TensorFlow Lite
converter. I'll create an instance of the TFLiteConverter and set it to perform integer quantization without any lossy
scaling factors by using the default range [0, 255]. This means input types will be float32 during conversion but mapped
to uint8 at inference.

I also need to specify the supported operators. Including both TFLITE_BUILTINS and TFLITE_BUILTINS_INT8 ensures
compatibility with integer operations, which is crucial for TinyML deployment on microcontrollers.

After converting the model, I'll save it using tf.lite.KerasModel.save() to the specified path. Finally, I'll add a print
statement to confirm where the model was saved, which helps in verifying success.

I should make sure that the code is clean and executable without any errors. Also, naming variables clearly will reflect
the quantization process as per the user's requirements.


</think\>

```python
import tensorflow as tf
import os

# Load the original Keras model
model = tf.keras.load_model('models/fruit_to_emoji/og_model/model.keras')

# Convert the model to TFLite with 8-bit integer quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.quantize_and_retain_float32_weights = True  # Retains float32 weights for accurate conversion

# Set input and output types to 8-bit integers
converter.inference_input_type = tf.uint8  # Input type is uint8
converter.inference_output_type = tf.uint8  # Output type is uint8

# Include built-in operators and integer operations in the support set
converter.supported_ops = [
    tf.lite.OpsSet.TFLITE_BUILTINS,
    tf.lite.OpsSet.TFLITE_BUILTINS_INT8
]

# Convert and save the model
tflite_model = converter.convert()
with open('models/fruit_to_emoji/tflite_model/model_quant.tflite', 'wb') as f:
    f.write(tflite_model)

print(f"Model saved to {os.path.abspath('models/fruit_to_emoji/tflite_model/model_quant.tflite')}")
```
````

Figure 18: An example output of `deepseek-r1:14b`.

Stages	Overall Token				Input Token				Output Token			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	31304	29202	57981	13742	18742	17441	35044	7925	12562	11416	22937	5391
MC	8589	9334	11289	1248	4991	5462	6716	381	3598	3627	5213	867

Table 46: Token consumption details regarding Overall, Input, and Output in three stages. Model: deepseek-r1:14b, Date: 01.23

Stages	Execution Time (s)			Token Consumption			Success Rate
	Ave	Max	Min	Ave	Max	Min	
DP	214.34	388.05	92.57	31304	57981	13742	0.267
MC	80.23	108.47	19.11	8589	11289	1248	0.333

Table 47: Performance metrics including execution time, token consumption, and success rate in three stages. Model: deepseek-r1:14b, Date: 01.23

Stages	Overall Cost (USD cents)				Input Cost (USD cents)				Output Cost (USD cents)			
	Ave	Med	Max	Min	Ave	Med	Max	Min	Ave	Med	Max	Min
DP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 48: Cost details regarding Overall, Input, and Output in three stages. Model: deepseek-r1:14b, Date: 01.23

Stages	Ave Execution Time (s)			Ave Total Tokens			Success Rate
	Overall	Pass	Fail	Overall	Pass	Fail	
DP	214.34	286.95	187.93	31304	40695	27890	0.267
MC	80.23	63.15	88.77	8589	6435	9667	0.333

Table 49: Performance metrics of passed and failed runs,including execution time, token consumption, and success rate in three stages. Model: deepseek-r1:14b, Date: 01.23

Stages	Ave	Med	Max	Min
DP	214.34	194.73	388.05	92.57
MC	80.23	83.63	108.47	19.11

Table 50: Time consumption (seconds) in three stages. Model: deepseek-r1:14b, Date: 01.23

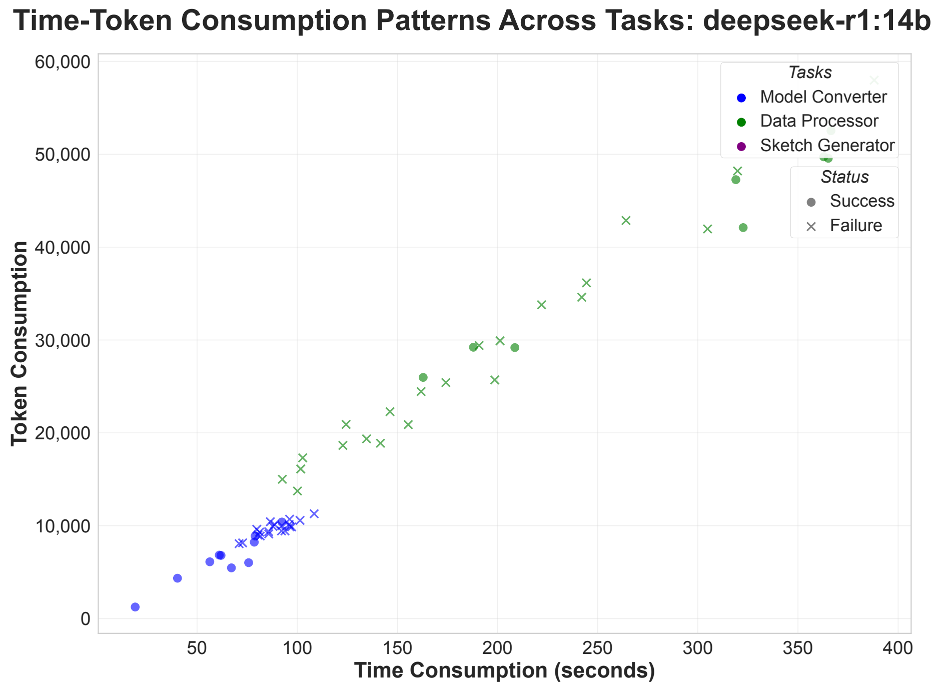


Figure 19: Distribution of time and token consumption in three tasks. Model: deepseek-r1:14b, Date: 01.23.

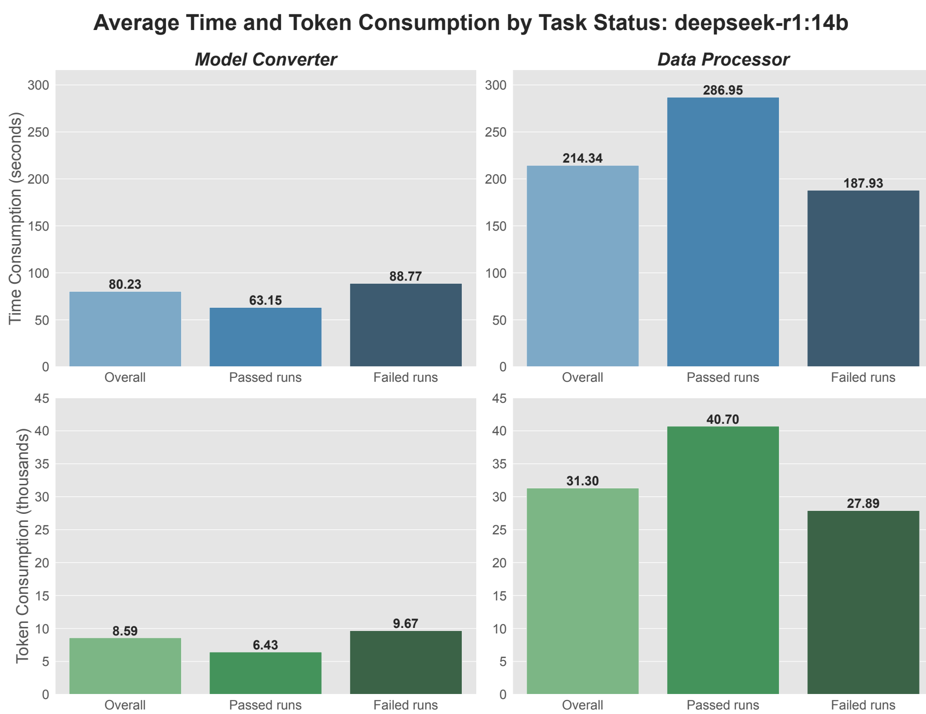


Figure 20: Average Time and Token Consumption by Task Status. Model: deepseek-r1:14b, Date: 01.23. *The average time (top) and token consumption (bottom) are compared across three stages, broken down by overall performance (considering all runs), performance of only successful runs, and of only the failed runs.*

1.12 Discussion: 2025.02.04, Output format handling issues with open-source models

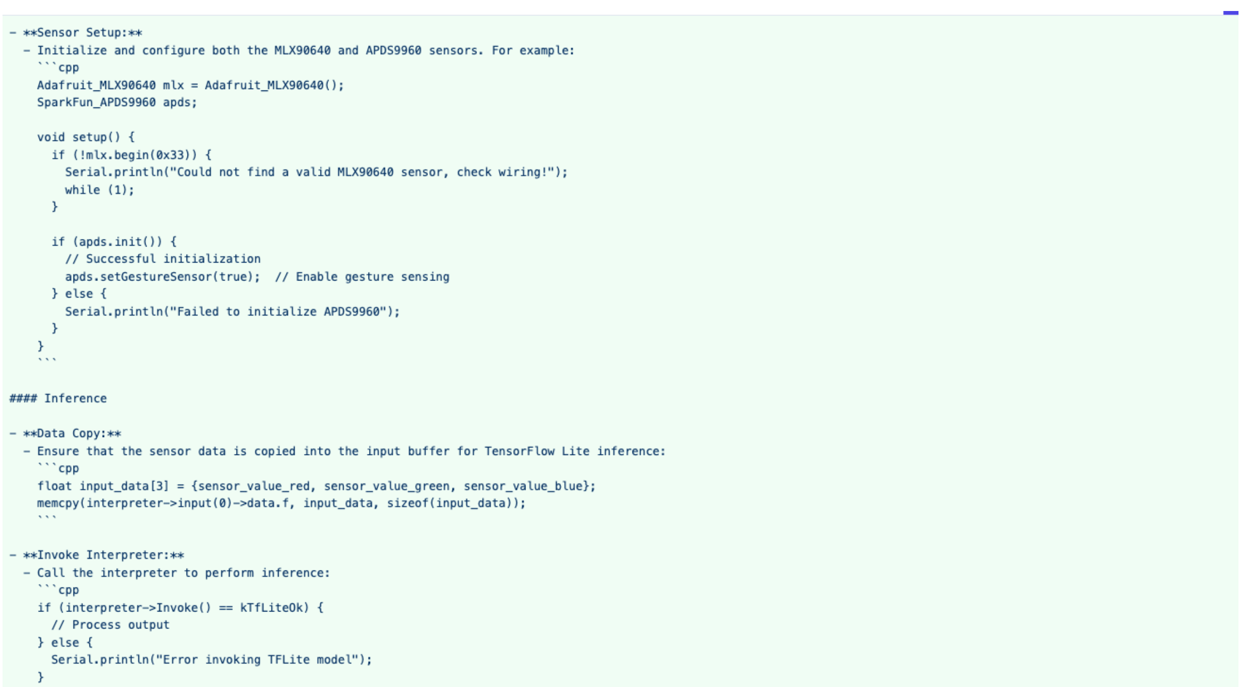
Open-source models' output can break the workflow by making faults in the middle and completely misleading the follow-up steps in the flow by **case 1).** incorrect/unparsable format of output (especially when i ask for merely code blocks while it outputs with explaninary text), **case 2).** wrong information or important information skipped, with right format, causing false positive results, **case 3).** not following the instructions in the prompt, not understanding the output/dataset path so that modifications happen in the directory it shouldnt touch. The SG stage is especially vulnerable to the issue case 2. DP is sometimes confused by issue case 3.

Here are the cases with examples:

1.12.1 Case 1 - incorrect/unparsable format of output:

This is very common in all stages, especially in DP and SG, because they have complex/multi-step code generation. E.g., when repeatedly debuging one step kept getting errors, the LLM tends to instrut me something about its code, or they output only the code snippets of the needed correction instead of outputting full code.

It's fine to output explanation text along with (before or after) the code block containing complete code, because the output parsing will pick out the `'''python/cpp/ino/json '''` code block and run it. What's not dealable with is **1)** mixing explanation text with code blocks, when they are alternate, code snippet + explanation by code snippet + explanation, as shown in the example 21; or similarly, giving text as the majority, code snippets as the exmaple of its explanation, dispersive and incomplete code blocks, as examplified in 22; **2)** Giving multiple solutions in the code, telling user to choose one accrodingly. Typically with alternative code pieces (method invocations, etc.) commented out, as shown in 23. **To conclude, it doesnt work when there is no single one code block containing complete code in the output, while this is always required in the prompt.**



````
- **Sensor Setup:**
  - Initialize and configure both the MLX90640 and APDS9960 sensors. For example:
    ```cpp
    Adafruit_MLX90640 mlx = Adafruit_MLX90640();
    SparkFun_APDS9960 apds;

    void setup() {
      if (!mlx.begin(0x33)) {
        Serial.println("Could not find a valid MLX90640 sensor, check wiring!");
        while (1);
      }

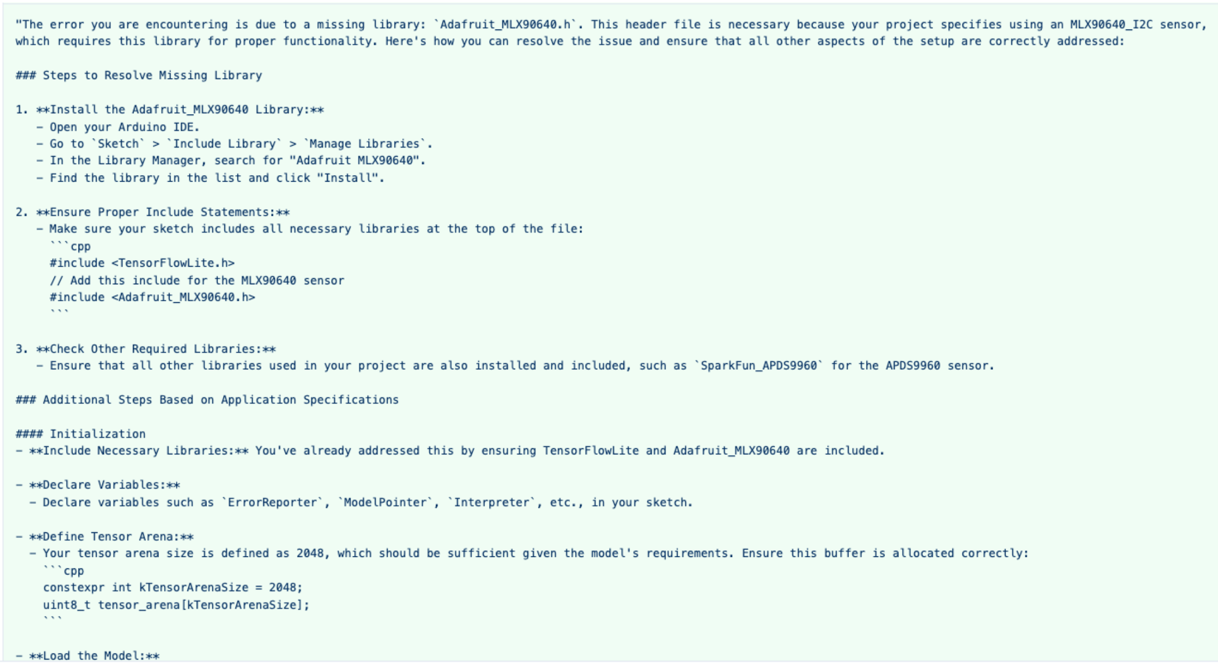
      if (apds.init()) {
        // Successful initialization
        apds.setGestureSensor(true);  // Enable gesture sensing
      } else {
        Serial.println("Failed to initialize APDS9960");
      }
    }
    ```

#### Inference

- **Data Copy:**
  - Ensure that the sensor data is copied into the input buffer for TensorFlow Lite inference:
    ```cpp
    float input_data[3] = {sensor_value_red, sensor_value_green, sensor_value_blue};
    memcpy(interpreter->input(0)->data.f, input_data, sizeof(input_data));
    ```

- **Invoke Interpreter:**
  - Call the interpreter to perform inference:
    ```cpp
    if (interpreter->Invoke() == kTfLiteOk) {
      // Process output
    } else {
      Serial.println("Error invoking TFLite model");
    }
    ```
````

Figure 21: Example of code and text being alternate.



```
"The error you are encountering is due to a missing library: `Adafruit_MLX90640.h`. This header file is necessary because your project specifies using an MLX90640_I2C sensor, which requires this library for proper functionality. Here's how you can resolve the issue and ensure that all other aspects of the setup are correctly addressed:

### Steps to Resolve Missing Library

1. **Install the Adafruit_MLX90640 Library:**
   - Open your Arduino IDE.
   - Go to `Sketch` > `Include Library` > `Manage Libraries`.
   - In the Library Manager, search for "Adafruit MLX90640".
   - Find the library in the list and click "Install".

2. **Ensure Proper Include Statements:**
   - Make sure your sketch includes all necessary libraries at the top of the file:
     ```cpp
     #include <TensorFlowLite.h>
     // Add this include for the MLX90640 sensor
     #include <Adafruit_MLX90640.h>
     ```

3. **Check Other Required Libraries:**
   - Ensure that all other libraries used in your project are also installed and included, such as `SparkFun_APDS9960` for the APDS9960 sensor.

Additional Steps Based on Application Specifications

Initialization
- **Include Necessary Libraries:** You've already addressed this by ensuring TensorFlowLite and Adafruit_MLX90640 are included.

- **Declare Variables:**
  - Declare variables such as `ErrorReporter`, `ModelPointer`, `Interpreter`, etc., in your sketch.

- **Define Tensor Arena:**
  - Your tensor arena size is defined as 2048, which should be sufficient given the model's requirements. Ensure this buffer is allocated correctly:
    ```cpp
    constexpr int kTensorArenaSize = 2048;
    uint8_t tensor_arena[kTensorArenaSize];
    ```

- **Load the Model:**
```

Figure 22: Example of code given under/inside explaninary text.

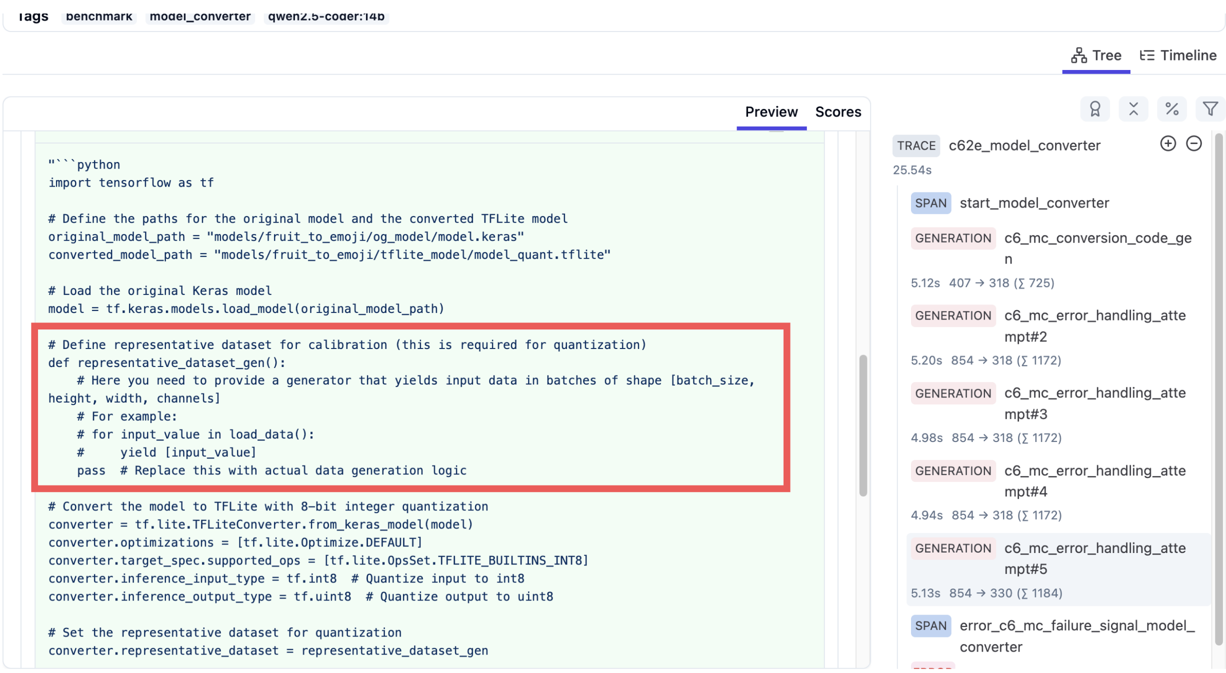


Figure 23: Example of given solution examples which requires user to choose one.

1.12.2 Case 2 - bad code with missing operations passes the test in a false positive way:

This was actually a bug in the error handing for specification filing and now it's fixed: *I was mentioning that when constructing the skecth specification for skecth generation, bad spec list with missing information leads to something-else sketch code, and that allows false positve sketch generation (under wrong or faulty specification, the returen code is compilable and passing our test, but the skecth is not practically usable, e.g.,* *a sketch simply invokes color senser and has nothing to do with the machine leaning model).*

But the problems with usability of compilable sketch still exists. I found that some of the compiled `.ino` files indeed has no response when flashed to Arduino, and this needs to be attach importance to and addressed. I think this is where we would use our testbed.

This also happens in DP, due to it's multi-step and long process natures of one whole task. For example, the image 24 shows the case that the code reads the given path correctly, but doesnt save the processed dataset, while reporting the dataset is updated. This causes false positive in the current step, and breaking the follow-up steps.The next step would seek for a path that doesn't exist, and the whole flow would be broken.

assistant

````
"```python
import pandas as pd

# Read the dataset
normalized_fruit_data_path = "./data/fruit_to_emoji/playground/normalized_fruit_data.csv"
df = pd.read_csv(normalized_fruit_data_path)

# Analyze class distribution
class_distribution = df['Fruit'].value_counts()

# Print the class distribution
print("Class Distribution:")
print(class_distribution)
```
```json
{
  "updated_class_distribution_path": "./data/fruit_to_emoji/playground/class_distribution.csv"
}
```"
````

Figure 24: Example of unsaved but reported data processing, causing false positive in the current step, and breaking the follow-up steps.

1.12.3 Case 3 - messing up directories:

All the data processing related mediate artefacts, including `*.csv` files and folders like `processed` should only be placed under `data/fruit_to_emoji/playground`. And the artefacts about model conversion like `*.tflite` should be under `models/fruit_to_emoji/tflite_model`. As shown in the image 25, all the mediate folders and files before which there's the red bar are located in wrong plcaes. They should either be under `data/fruit_to_emoji/playground` which is rectangled in green, or `models/fruit_to_emoji/tflite_model`.

This also includes the naming of the files, especially the converted model, the name is predefined as `model_quant_int8.tflite` in our code, but e.g. under the second red bar in 25, it's named as `converted_model.tflite`, as well as put in the wrong directory.

This happens directly because the path variables defined in the output code do not match the specificatio in the prompt.
````

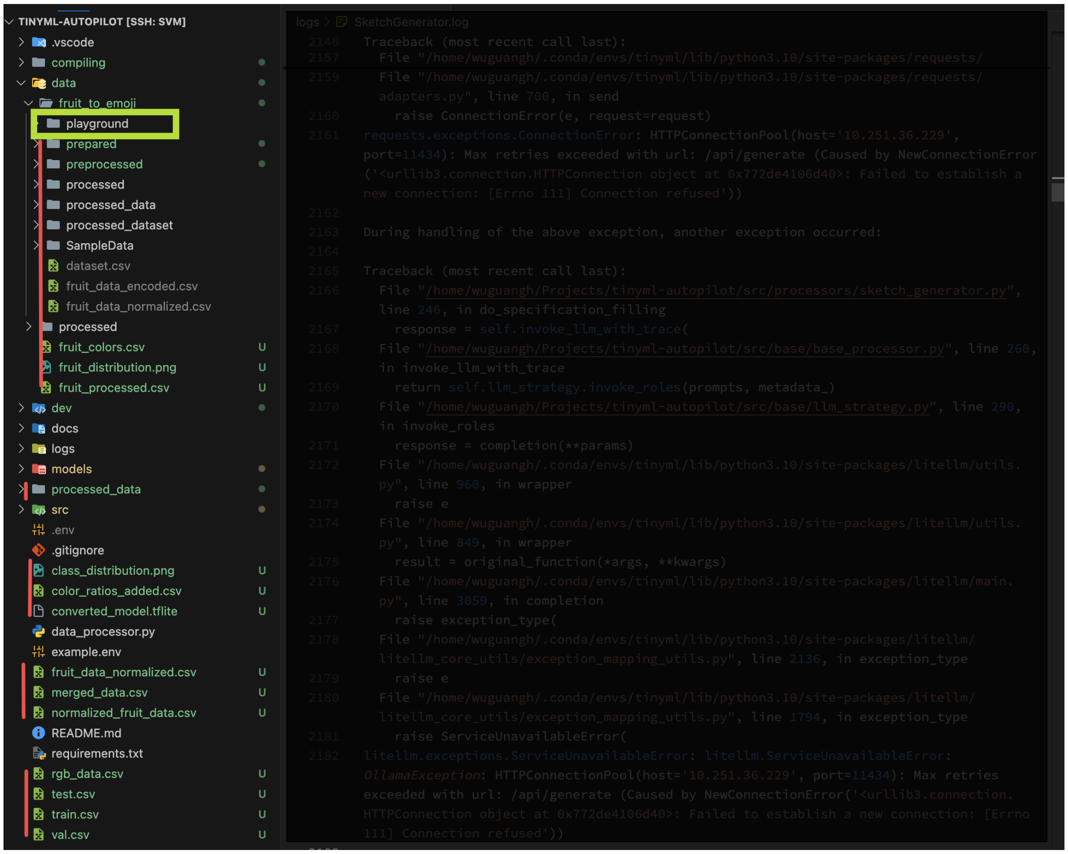


Figure 25: Results of directory mess-up.

2 Future Extensions

- Add library installation step to sketch generation.
  Each code generation output can include a section, especially for imported libraries. The local executor installs those libraries for arduino-cli before trying compilation.
- This is very interesting: https://huggingface.co/blog/unified-tool-use. I will write something about this in the current paper.

2.1 Extensions Mentioned in the Thesis

- **Expanding Lifecycle Coverage**: Expanding the system to cover additional stages of the TinyML lifecycle, such as model designing and training, would provide a more comprehensive automation solution.
- **Expanding Model and Hardware Coverage**: Testing this framework with a wider range of TinyML models and hardware platforms to assess its versatility and identify potential improvements.
- **LLM Comparison**: Evaluating the performance of this framework with different LLMs to understand how the choice of LLM impacts the system's effectiveness.
- **Performance Benchmarking**: Conducting comprehensive benchmarks comparing the LLM-powered approach to traditional development methods in terms of development time, code quality, and application performance would provide valuable insights into the system's practical benefits, and make the proposal more trustable.

- **Qualitative Analysis**: Conducting a formal qualitative evaluation involving a questionnaire provided to different individuals for testing the proposed solution and comparing it with the traditional methods. This would provide valuable insights into real-world applicability.
- **Improving Reliability**: Enhancing the success rate of code generation, particularly for sketch generation. This can be done by refining prompt engineering to have finer control of LLM's behavior.
- **Specialized Fine-Tuning**: Fine-tuning LLMs specifically for TinyML tasks could improve LLM's performance and reliability in this domain.
- **Integration with Traditional Tools**: Combining LLM-powered automation with traditional TinyML tools could leverage the strengths of both approaches.
- **User Interface Development**: Creating intuitive interfaces for interacting with the LLM system could facilitate the usage of this system.

3 Bibliography List

[1] [2] [3]

References

[1] Z. Englhardt, R. Li, D. Nissanka, Z. Zhang, G. Narayanswamy, J. Breda, X. Liu, S. Patel, and V. Iyer, “Exploring and Characterizing Large Language Models For Embedded System Development and Debugging,” Nov. 2023.

[2] S. Dou, H. Jia, S. Wu, H. Zheng, W. Zhou, M. Wu, M. Chai, J. Fan, C. Huang, Y. Tao, Y. Liu, E. Zhou, M. Zhang, Y. Zhou, Y. Wu, R. Zheng, M. Wen, R. Weng, J. Wang, X. Cai, T. Gui, X. Qiu, Q. Zhang, and X. Huang, “What’s Wrong with Your Code Generated by Large Language Models? An Extensive Study,” July 2024.

[3] S. Gao, J. Dwivedi-Yu, P. Yu, X. E. Tan, R. Pasunuru, O. Golovneva, K. Sinha, A. Celikyilmaz, A. Bosselut, and T. Wang, “Efficient Tool Use with Chain-of-Abstraction Reasoning,” Feb. 2024.