

A INCREMENTAL VERSION OF LIBRARIAN

We support sharing library components across clusters through the below incremental version of our algorithm. It processes clusters sequentially, rather than in parallel, and its output can depend on the ordering imposed upon the clusters.

$$\begin{aligned}
 \mathcal{L}_0 &= \emptyset \\
 \{\rho'_n\} &= \bigcup_{c \in \text{CLUSTER}(\{\rho_n\})} \{\rho'_{c,i}\}_{i=1}^{|c|} \\
 (\Delta\mathcal{L}_c, \{\rho'_{c,i}\}_{i=1}^{|c|}) &= \arg \min_{(\Delta\mathcal{L}, \{\rho'_i\}) \in \text{SAMPLE}_k(c; \mathcal{L}_{t-1})} \ell(\mathcal{L}_{t-1} \cup \Delta\mathcal{L}, \{\rho'_i\}_{i=1}^{|c|}) \quad \forall c \in \text{CLUSTER}(\{\rho_n\}) \\
 \mathcal{L}_t &= \mathcal{L}_{t-1} \cup \Delta\mathcal{L}_t, \quad \mathcal{L}^* = \mathcal{L}_{|\text{CLUSTER}(\{\rho_n\})|}
 \end{aligned}$$

B ALGORITHM

Algorithm 1 Refactoring Specialized Programs into a Joint Library

Require: Set of independent, specialized programs $P_{\text{initial}} = \{\rho_1, \rho_2, \dots, \rho_n\}$

Require: Sample Budget K

Ensure: Joint library $\mathcal{L}_{\text{final}}$ and set of refactored programs P_{final}

```

1:  $\mathcal{C} \leftarrow \text{Cluster}(P_{\text{initial}})$ 
2:  $\mathcal{L}_{\text{final}} \leftarrow \emptyset, P_{\text{final}} \leftarrow \emptyset$ 
3: for all cluster  $c \in \mathcal{C}$  do                                     ▷ Each cluster independently
4:    $T_C \leftarrow \text{GroupIntoTuples}(c)$                                ▷ Get tuples for each cluster
5:   for all tuples  $\tau \in T_C$  do
6:      $\{f_{\text{retrieved}}\} \leftarrow \text{RetrieveRelevantFromLibrary}(\mathcal{L}, \tau)$ 
7:      $S \leftarrow \emptyset$ 
8:     for  $i = 1$  to  $K$  do                                             ▷ Sample  $k$  times
9:        $(\{f_{\text{new},i}\}, \{\rho'_i\}) \leftarrow \text{SAMPLE}(f_{\text{retrieved}}, \tau)$ 
10:       $S \leftarrow S \cup \{(\{f_{\text{new},i}\}, \{\rho'_i\})\}$ 
11:    end for
12:     $(f_{\text{best}}, \{\rho'_{\text{best}}\}) \leftarrow \text{RerankAndSelectBest}(S, \ell(\cdot))$    ▷ Rerank using objective
13:     $\mathcal{L}_{\text{final}} \cup \{f_{\text{best}}\}$ 
14:     $P_{\text{final}} \cup \{\rho'_{\text{best}}\}$ 
15:  end for
16: end for
17: return  $\mathcal{L}_{\text{final}}, P_{\text{final}}$ 

```

C EXPERIMENTAL SETUP

Grouping Programs into Collections To facilitate parallel application of LIBRARIAN and manage the dataset scale, we assume that semantically distant files will have minimal overlap in their optimal library functions. Therefore, our overall approach partitions the dataset into disjoint collections through clustering.

For **CodeContests**, these collections are constructed from an initial corpus of $\sim 9\text{k}$ problems with Python solutions: We first filter these files, removing those whose selected canonical solution is under 10 lines (minimal refactoring potential). For the remaining 4596 solutions we use a language model to generate textual descriptions of canonical solutions—emphasizing reusable components—which are embedded using OpenAI’s text-embedding-ada-002.

Agglomerative Clustering (Ward Jr, 1963) is subsequently applied to these embeddings to partition the files into a predefined number of initial clusters, in our case 120. To create uniformly sized experimental units, we subsample each such cluster to form collections of 30 files. This collection

size was empirically chosen because it balanced between the runtime of LIBRARIAN without limiting compression. We select 10 collections that we then use to evaluate our methods.

For Transformers, since the number of models is on the lower end, we manually chose a set of popular LLM / VLM models and passed them to the agent in collections of 5 code sources.

REGAL Baselines. To evaluate the ability of our libraries to support reuse on new problems, we turn to the program synthesis tasks used in REGAL, where learned libraries are added to help the program synthesizer. We evaluate on the two domains published by the authors, Logo and Date. Because our clustering is inspired by REGAL but adds additional complexity, for fair comparison, we keep their setup the same and only augment the training using sample + MDL rerank procedure described in Section 5.

Code Contests. To evaluate LIBRARIAN on refactoring Code Contests we select 6 collections of 30 files (problems). In each collection we group the problems into tuples of size 3. We set the sample budget to be $K = 8$, since our ablations show that with larger K we discover better libraries 2. We use the MDL objective for rankings. The model used for sampling is OpenAI’s o4-mini (OpenAI, 2024). To obtain MDL scores we use Qwen 2.5 7B Instruct (Qwen et al., 2025) as a balance between quality, speed, and cost.

Code Agents on Transformers and Diffusers Repositories. To fairly evaluate performance on the task by state-of-the-art systems, we use coding agents that advertise long-context ability to reason about, write, and refactor code repositories. Specifically, we use Claude Code (CI) (Anthropic, 2025) which uses the Opus 4.1.

We test whether code agents can refactor collections of code sources autonomously, without human intervention. Refactoring repositories with code agents involves planning and iterative (re-)implementation and testing. Code agents are prompted to perform each of these steps, with feedback from the unit tests. Agents must run and repair unit tests autonomously. We run coding agents multiple times per task, logging their progress in checklists stored in text files.

The instruction provided to human evaluators is as follows:

```

1
2
3 ## 1. Materials Provided
4
5 You will be given a set of files for each example case:
6
7 * **original_programs.py**: This file contains a set of 3 distinct Python programs, each presented with its
8   corresponding problem description/query. This represents the "before" state.
9 * **v1.py**: This file presents the first refactoring approach. It includes:
10   * The 3 refactored versions of the original programs.
11   * A "library" section (e.g., 'codebank.py' or inline) containing helper functions. These helper functions
12     might be retrieved from an existing common library or newly created during this refactoring.
13   * Either the retrieved or the new helper function sections may be _empty_, in case no programs existed in
14     the codebank at the time or if no need helper functions were created by the LLM.
15 * **v2.py**: This file presents the second, alternative refactoring approach. Similar to 'refactoring_v1.py',
16   it includes:
17   * The 3 refactored versions of the original programs (using a different strategy than v1).
18   * A "library" section with its own set of helper functions.
19
20 **NOTE**: both refactorings had accuracy at least as good as the original programs.
21
22 ## 2. Your Task
23
24 Your primary task is to:
25
26 1. **Review** the 'original_programs.py' to understand the initial code and the problems being solved.
27 2. **Analyze** both 'refactoring_v1.py' and 'refactoring_v2.py'. Pay close attention to how the original
28   programs have been restructured and what functionalities have been extracted into their respective
29   libraries.
30 3. **Decide** which refactoring (Version 1 or Version 2) you believe is "better,"** based on the evaluation
31   criteria provided below (or your own criteria!).
32
33 ## 3. Evaluation Criteria: What to Consider for Your Choice
34
35 When comparing 'refactoring_v1.py' and 'refactoring_v2.py', please *consider* the following aspects to inform
36 your choice. The "better" refactoring should ideally excel in these areas:
37
38 > Most importantly, make sure that the extracted functions are **actually reusable and not too specific.** If
39   the main programs are short, the refactoring is not immediately "better"! Try to think whether the
40   extracted functions could actually be used in a different program down the line.
41
42 * **Reusability of Helper Functions **:
43   * **Generality**: Are the new helper functions general-purpose and potentially useful for *other,
44     different* programs and problems beyond the three presented?
45   * **Reuse**: How much were existing helper functions reused?
46   * **Specificity**: Are the functions too specialized to the current set of problems, limiting their
47     broader applicability? _Avoid_ functions that are essentially just the original program broken out into a
48     "helper._"
49   * Composability
50 * **Maintainability**:
51   * Readability & Understandability
52   * Ease of Modification
53   * Separation of Concerns
54
55 ## 4. What NOT to Focus On:
56
57 * **Comments**: Please disregard the presence or absence of comments in the code for this evaluation. These
58   are superficially generated by LLMs in some occasions and could be added manually after with a single
59   pass.
60 * **Minor Stylistic Differences**: Do not focus on trivial differences in variable naming or formatting,
61   unless they significantly impact readability or understanding.
62
63 ## 5. How to Provide Your Feedback
64
65 For each example case, please provide:
66
67 1. **Your Preferred Version**: (e.g., "Version 1" or "Version 2")

```

Listing 1: Human Evaluation Instruction

D ASYMPTOTIC BEHAVIOR RESULTS

Here we include the full graph of asymptotic behavior of all four scoring metrics (MDL, tokens, MI and CC), reporting the raw library metrics and the metric ratios. We can see that refactored programs have higher CC than the baseline and that optimizing CC as an objective does not decrease the other metrics. MDL performs best on total raw library usage as well as on average times a library function is used in the refactorings. MI ends up optimizing the number of library functions the best, but their reusability is below the average for the refactorings. Optimizing for tokens produces smaller libraries with less usage per function compared to optimizing MDL.

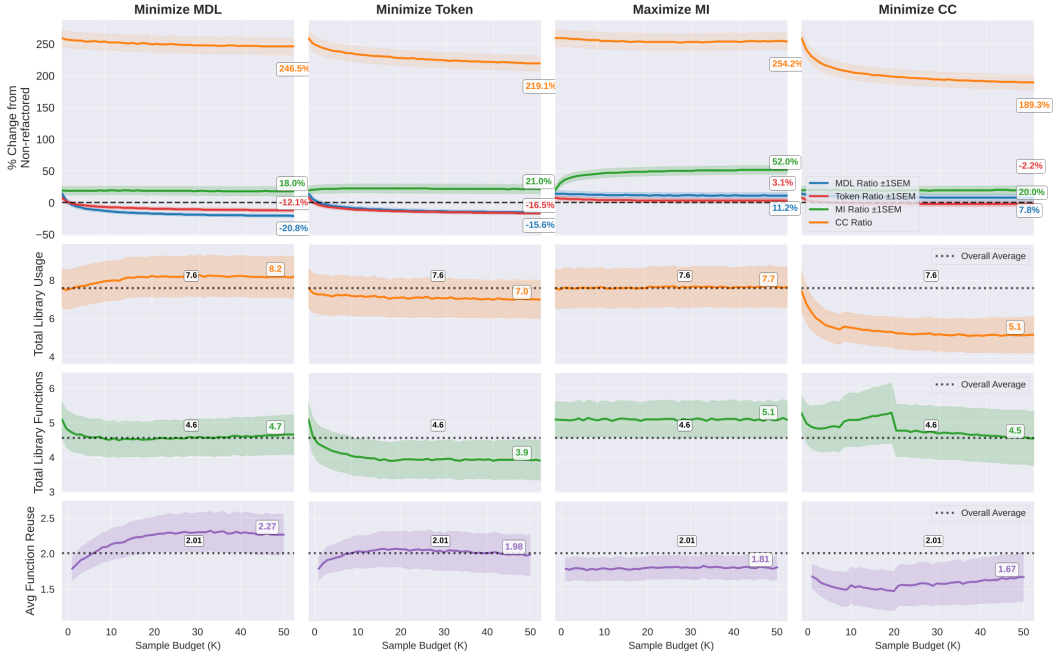


Figure 7: Asymptotic behavior of metrics for scoring libraries and refactorings (columns) varying refactoring budget (horizontal axes).

E HUMAN STUDY DETAILS

We ran a user study with 12 participants where each participant had to judge 10 refactoring pairs. Each pair was comparing two out of three metrics, MDL, tokens, and MI. We showed the participants the original programs, as well as both of the refactoring versions (including the programs and the learned libraries). The participants were able to choose either version or say that the two refactorings are almost the same.

To quantify pairwise preferences between refactoring metrics, we employed the Bradley-Terry model, a standard framework for analyzing paired comparison data. We fit the model using maximum likelihood estimation with mutual information (MI) as the reference category ($\pi_M I = 1$). To address potential noise in human judgments, we applied consensus-based filtering with a 75% threshold, retaining all responses for comparisons with low consensus (indicating genuine ambiguity) while excluding minority responses on high-consensus comparisons where the majority preference likely indicates the correct judgment. This conservative approach preserved 94.2% of responses while strengthening the statistical evidence for metric preferences, with MDL significantly outperforming MI ($p = 0.67$, 95% Confidence Interval: $[0.56, 0.78]$). Even without the filtering MDL preference over MI was statistically significant.

F BEST@K COMPRESSION IS A U-STATISTIC

We wish to estimate the expected compression ratio achieved by our *sample + rerank* method, which samples k candidate refactorings, discards any that do not pass the tests, and selects the one with the lowest score (total log-prob).

Background on U-Statistics. Let $Z_1, \dots, Z_n \stackrel{\text{i.i.d.}}{\sim} F$. For a symmetric function $h : \mathcal{Z}^k \rightarrow \mathbb{R}$, the *U-statistic of order k* is defined as

$$U_n = \binom{n}{k}^{-1} \sum_{1 \leq i_1 < \dots < i_k \leq n} h(Z_{i_1}, \dots, Z_{i_k}). \quad (6)$$

By construction,

$$\mathbb{E}[U_n] = \mathbb{E}[h(Z_1, \dots, Z_k)], \quad (7)$$

so U_n is an unbiased estimator of the population quantity $\theta = \mathbb{E}[h(Z_1, \dots, Z_k)]$.

Application to Best@k Compression. Let each valid refactoring be a pair $Z = (S, C)$, where S is the score and C is the compression ratio. Define the symmetric function

$$h_k(z_1, \dots, z_k) = C_{j^*}, \quad j^* = \arg \min_{1 \leq j \leq k} S_j, \quad (8)$$

the compression ratio of the lowest-score refactoring among k draws. The population target is then

$$\theta_k = \mathbb{E}[h_k(Z_1, \dots, Z_k)]. \quad (9)$$

Given n valid samples, our estimator is

$$\hat{\theta}_k = \binom{n}{k}^{-1} \sum_{1 \leq i_1 < \dots < i_k \leq n} h_k(Z_{i_1}, \dots, Z_{i_k}). \quad (10)$$

Proposition. $\hat{\theta}_k$ is a U-statistic of order k with function h_k , and hence an unbiased estimator of θ_k .

Proof. (1) *Symmetry of the function h_k .* h_k selects the compression associated with the lowest score among its k arguments. Permuting the inputs does not affect this outcome (ties can be resolved with a fixed, permutation-invariant rule). Thus h_k is symmetric.

(2) *U-statistic form.* By definition, a U-statistic of order k with kernel h_k is

$$U_n = \binom{n}{k}^{-1} \sum_{1 \leq i_1 < \dots < i_k \leq n} h_k(Z_{i_1}, \dots, Z_{i_k}),$$

which matches $\hat{\theta}_k$ exactly.

Therefore, $\hat{\theta}_k$ is a U-statistic of order k . By the unbiasedness property of U-statistics,

$$\mathbb{E}[\hat{\theta}_k] = \theta_k.$$

□

Thus, our reported *best@k compression curves* provide unbiased estimates of the expected performance of the *sample + rerank* method.

G CLUSTERING ANALYSIS: CODECONTESTS

We analyze the coherence of the clusters underlying collections in **MINICODE-CodeContests**. In particular, we compare clustering based on o4-mini generated file descriptions against task descriptions. Since task descriptions in competition coding problems are designed to hide the algorithmic approach needed to solve problem, we expect that clusters based on file descriptions are more coherent. We use Normalized Tag Instance Entropy and Herfindahl-Hirschman Index to evaluate clusterings. Figure 8 shows our clustering approach yields more thematically coherent clusters, evidenced by achieving lower entropy and higher HHI values across the entire tested range of N . We provide definitions of our measures below.

G.1 COLLECTION COHERENCE MEASURES

We use two measures to evaluate the thematic coherence of collections: Good collections should group files with a (1) concentrated and (2) identifiable set of shared *conceptual tags*, which for CodeContests are provided as ground truth (trees, graphs, etc.).

We provide the full definitions of the collection coherence measures below.

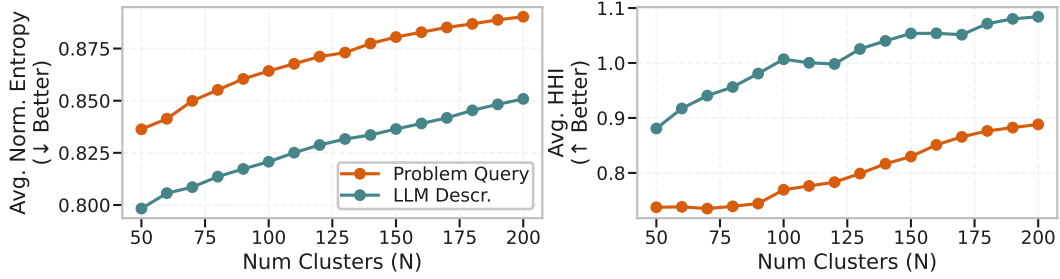


Figure 8: Clustering analysis of 4,596 Code Contest problems, comparing the thematic coherence of clusters formed using our proposed method versus REGAL-style clustering.

Normalized Tag Instance Entropy: This measures the concentration of tag *instances* within a collection C . Let p_i be the proportion of the i -th unique tag type among all tag instances in C , and D_C be the number of distinct tag types in C . If $D_C > 1$, the normalized entropy H_N is defined as:

$$H_N = -\frac{\sum_{i=1}^{D_C} p_i \log_2 p_i}{\log_2 D_C} \quad (11)$$

If $D_C \leq 1$, then $H_N = 0$. Lower H_N (closer to 0) indicates higher thematic purity, meaning fewer tag types dominate the bulk of tag mentions.

Herfindahl-Hirschman Index (HHI) for Problem Presence: This measures tag concentration across distinct *problems* in a cluster C . Let s_t be the proportion of problems in C that include tag t (a problem contributes to s_t if t is one of its unique tags). A higher HHI signifies that the problems are collectively characterized by a smaller, more focused set of tags.

$$\text{HHI} = \sum_{t \in \text{Tags}(C)} s_t^2 \quad (12)$$

where $\text{Tags}(C)$ represents the set of unique tags present in cluster C .

H BENCHMARK COMPARISON

We compare our benchmark, MINICODE, to similar benchmarks in Table 4. We define creativity and design as the need to explore diverse solutions in order to find the best solution possible. For example, optimizing for program correctness alone does not require exploring a large solutions space, whereas optimizing a program for speed would. In the case of compressing large files, we must explore the large space of shared abstractions afforded by libraries in order to maximize compression.

Table 4: Comparison of Code Benchmarks

Benchmark	Creativity/Design	Scale
SWE-bench (Jimenez et al., 2024)	Low	Repository
Commit-0 (Zhao et al., 2025)	Medium	Repository
RefactorBench (Gautam et al., 2025)	Low	File
ECCO (Waghjale et al., 2024)	High	Function
KernelBench (Ouyang et al., 2025)	High	Function
MINICODE(Ours)	High	Repository

I FULL MINICODE CODECONTESTS RESULTS

We present the full agent scores for the CodeContests split in Table 5. The results are given both for each cluster of code sources, as well as averaged across clusters.

Cluster	Agent	Tokens	CC	Pass %	MDL	MDL %
0	original	9088	95	80.3	11745.85	100.0
	sonnet 3.7	18114	176	87.0	15005.18	127.7
	sonnet 4	11121	138	80.3	9901.53	84.3
	codex-mini	9321	95	80.3	9990.74	85.1
1	original	12531	255	89.7	13431.86	100.0
	sonnet 3.7	10470	239	96.7	8933.65	66.5
	sonnet 4	11325	298	96.7	8214.42	61.2
	codex-mini	12762	255	89.7	11798.73	87.8
2	original	14087	376	89.0	15012.77	100.0
	sonnet 3.7	17345	429	91.3	13145.02	87.6
	sonnet 4	14270	356	93.0	10522.66	70.1
	codex-mini	14318	376	89.0	13273.81	88.4
3	original	14261	246	90.3	13348.82	100.0
	sonnet 3.7	20749	241	97.7	15859.02	118.8
	sonnet 4	13433	197	80.7	11937.04	89.4
	codex-mini	14495	246	90.3	11616.41	87.0
4	original	17693	336	80.7	14665.16	100.0
	sonnet 3.7	29860	358	100.0	20666.52	141.0
	sonnet 4	18684	352	82.0	12801.21	87.3
	codex-mini	17923	336	80.7	12902.09	88.0
5	original	12588	286	92.0	12790.11	100.0
	sonnet 3.7	10580	128	99.3	8435.12	65.9
	sonnet 4	10416	155	99.3	9167.85	71.7
	codex-mini	12819	286	92.0	11086.19	86.7
6	original	11020	131	54.3	13540.41	100.0
	sonnet 3.7	21747	502	88.0	19446.07	143.6
	sonnet 4	11177	143	57.3	10492.00	77.5
	codex-mini	11251	131	54.3	11651.65	86.1
7	original	12301	180	80.0	12393.73	100.0
	sonnet 3.7	16390	166	91.0	13371.59	107.9
	sonnet 4	11625	150	85.7	9304.25	75.1
	codex-mini	12534	180	80.0	10549.04	85.1
Avg	original	12946	238	82.0	13366.09	100.0
	sonnet 3.7	18157	280	93.9	14357.77	107.4
	sonnet 4	12756	224	84.4	10292.62	77.1
	codex-mini	13178	238	82.0	11608.58	86.8

Table 5: Comparison of the pass rate and compression metrics of the original files, Claude Sonnet 4 and codex-mini refactorings across CodeContests clusters.

J REFACTORING EXAMPLES OF LIBRARIAN ON CODE CONTESTS

J.1 EXAMPLE 1

In code snippets 3, 2, 5, 4 one example of 2 refactoring versions. Specifically, the versions are both passing at least as many test cases as the original and they have the biggest difference in MDL among all the sample refactorings for that tuple. Sample + rerank filtering selected refactoring V2. You can observe that refactoring V1 introduces some problem specific functions like `build_max_beauty_perm()`, while refactoring V2 sticks to more generally useful functions.

```

972 1 # ==== NEW HELPER FUNCTIONS ====
973 2 def compute_full_mask(i):
974 3     """Return mask of all 1s of the bit-length of i."""
975 4     return (1 << i.bit_length()) - 1
976 5
977 6 def build_max_beauty_perm(n):
978 7     """Build permutation of 0..n maximizing sum of i^p[i]."""
979 8     ans = [0] * (n + 1)
980 9     used = set()
981 10    for i in range(n, -1, -1):
982 11        if i in used:
983 12            continue
984 13        mask = compute_full_mask(i)
985 14        j = i ^ mask
986 15        ans[i], ans[j] = j, i
987 16        used.add(i)
988 17        used.add(j)
989 18    beauty = sum(i ^ ans[i] for i in range(n + 1))
990 19    return ans, beauty
991 20
992 21 def solve_xor_sum(u, v):
993 22     """
994 23     Find shortest array whose xor is u and sum is v.
995 24     Return list or None if impossible.
996 25     """
997 26     if u > v or (v - u) % 2:
998 27         return None
999 28     if u == v:
1000 29         return [] if u == 0 else [u]
1001 30     x = (v - u) // 2
1002 31     # try two elements
1003 32     if ((u + x) ^ x) == u:
1004 33         return [u + x, x]
1005 34     # fallback to three elements
1006 35     return [u, x, x]
1007 36
1008 37 def build_trie(keys):
1009 38     """
1010 39     Build a binary trie with counts for 30-bit numbers.
1011 40     Each node: [left_index, right_index, count].
1012 41     """
1013 42     tree = [[0, 0, 0]]
1014 43     for x in keys:
1015 44         now = 0
1016 45         tree[now][2] += 1
1017 46         for i in range(29, -1, -1):
1018 47             b = (x >> i) & 1
1019 48             if tree[now][b] == 0:
1020 49                 tree[now][b] = len(tree)
1021 50                 tree.append([0, 0, 0])
1022 51             now = tree[now][b]
1023 52             tree[now][2] += 1
1024 53     return tree
1025 54
1026 55 def trie_pop_min_xor(tree, x):
1027 56     """
1028 57     Pop one key from trie to minimize x^key and return that minimal xor.
1029 58     Decrements counts along the path.
1030 59     """
1031 60     now = 0
1032 61     res = 0
1033 62     for i in range(29, -1, -1):
1034 63         b = (x >> i) & 1
1035 64         nxt = tree[now][b]
1036 65         if nxt and tree[nxt][2] > 0:
1037 66             now = nxt
1038 67         else:
1039 68             now = tree[now][b ^ 1]
1040 69             res |= (1 << i)
1041 70             tree[now][2] -= 1
1042 71     return res

```

Listing 2: Version 1, New Helpers

```

1026 1
1027 2 ##### PROGRAM: node_16:cc_python_16 #####
1028 3
1028 4 from codebank import *
1029 5
1029 6 def main():
1030 7     import sys
1030 8     data = sys.stdin.readline()
1031 9     if not data:
1031 10         return
1032 11     n = int(data)
1033 12     perm, beauty = build_max_beauty_perm(n)
1033 13     print(beauty)
1034 14     print(*perm)
1035 15
1035 16 if __name__ == "__main__":
1036 17     main()
1037 18
1037 19 ##### PROGRAM: node_19:cc_python_19 #####
1038 20
1038 21 from codebank import *
1039 22
1039 23 def main():
1040 24     import sys
1040 25     data = sys.stdin.readline()
1041 26     n = int(data())
1042 27     A = list(map(int, data().split()))
1042 28     P = list(map(int, data().split()))
1043 29     trie = build_trie(P)
1044 30     O = [trie_pop_min_xor(trie, a) for a in A]
1044 31     print(*O)
1045 32
1045 33 if __name__ == "__main__":
1046 34     main()
1047 35
1047 36 ##### PROGRAM: node_25:cc_python_25 #####
1048 37
1048 38 from codebank import *
1049 39
1049 40 def main():
1050 41     import sys
1050 42     u, v = map(int, sys.stdin.readline().split())
1051 43     res = solve_xor_sum(u, v)
1052 44     if res is None:
1052 45         print(-1)
1053 46     else:
1053 47         print(len(res))
1054 48         if res:
1054 49             print(*res)
1055 50
1055 51 if __name__ == "__main__":
1056 52     main()

```

Listing 3: Version 1, Refactored Programs

```

1062 1 # === NEW HELPER FUNCTIONS ===
1063 2 def compute_complement(i):
1063 3     return i ^ ((1 << i.bit_length()) - 1)
1064 4
1065 5 def trie_add(trie, x, max_bit):
1065 6     trie[0][2] += 1
1066 7     now = 0
1066 8     for i in range(max_bit, -1, -1):
1067 9         bit = (x >> i) & 1
1067 10         if trie[now][bit] == 0:
1068 11             trie[now][bit] = len(trie)
1069 12             trie.append([0, 0, 0])
1069 13             now = trie[now][bit]
1070 14             trie[now][2] += 1
1071 15
1071 16 def trie_find_min_xor(trie, x, max_bit):
1072 17     now = 0
1072 18     ans = 0
1073 19     for i in range(max_bit, -1, -1):
1073 20         bit = (x >> i) & 1
1074 21         if trie[now][bit] and trie[trie[now][bit]][2] > 0:
1075 22             now = trie[now][bit]
1075 23         else:
1076 24             now = trie[now][bit ^ 1]
1076 25             ans |= (1 << i)
1077 26             trie[now][2] -= 1
1077 27     return ans
1078 27

```

Listing 4: Version 2, New Helpers

```

1080 1 ##### PROGRAM: node_16:cc_python_16 #####
1081 2
1082 3 from codebank import *
1083 4
1084 5 def main():
1085 6     import sys
1086 7     input = sys.stdin.readline
1087 8     n = int(input())
1088 9     ans = [-1] * (n + 1)
1089 10     for i in range(n, -1, -1):
1090 11         if ans[i] == -1:
1091 12             z = compute_complement(i)
1092 13             ans[i] = z
1093 14             ans[z] = i
1094 15     m = sum(i ^ ans[i] for i in range(n + 1))
1095 16     print(m)
1096 17     print(*ans)
1097 18
1098 19 if __name__ == "__main__":
1099 20     main()
1100 21
1101 22 ##### PROGRAM: node_19:cc_python_19 #####
1102 23
1103 24 from codebank import *
1104 25
1105 26 def main():
1106 27     import sys
1107 28     input = sys.stdin.readline
1108 29     n = int(input())
1109 30     A = list(map(int, input().split()))
1110 31     P = list(map(int, input().split()))
1111 32     max_bit = max(max(A, default=0), max(P, default=0)).bit_length() - 1
1112 33     trie = [[0, 0, 0]]
1113 34     for x in P:
1114 35         trie_add(trie, x, max_bit)
1115 36     res = [trie_find_min_xor(trie, x, max_bit) for x in A]
1116 37     print(*res)
1117 38
1118 39 if __name__ == "__main__":
1119 40     main()
1120 41
1121 42 ##### PROGRAM: node_25:cc_python_25 #####
1122 43
1123 44 from codebank import *
1124 45
1125 46 def main():
1126 47     u, v = map(int, input().split())
1127 48     if u > v or ((v - u) & 1):
1128 49         print(-1)
1129 50     elif u == 0 and v == 0:
1130 51         print(0)
1131 52     elif u == v:
1132 53         print(1)
1133 54         print(u)
1134 55     else:
1135 56         w = (v - u) // 2
1136 57         if (w & u) == 0:
1137 58             d = u + w
1138 59             print(2)
1139 60             print(d, w)
1140 61         else:
1141 62             print(3)
1142 63             print(u, w, w)
1143 64
1144 65 if __name__ == "__main__":
1145 66     main()

```

Listing 5: Version 2, Refactored Programs

J.2 EXAMPLE 2

In code snippets 7, 6, 9, 8 is another example of 2 refactorings where V1 was better according to LIBRARIAN. We can observe that V2 creates helper functions that are overly specific to the problem. You can see that refactoring V2 introduces overly specialized functions like `dijkstra_special()` or `compute_min_moves_opposite_parity()`. In comparison, refactoring V1 generates only general versions of these functions (e.g. `dijkstra()`).

```

1134 1
1135 2 # === NEW HELPER FUNCTIONS ===
1136 3 def read_ints():
1137 4     return list(map(int, input().split()))
1138 5
1139 6 def build_adj_undirected(n, edges):
1140 7     adj = [[] for _ in range(n)]
1141 8     for u, v, w in edges:
1142 9         adj[u].append((v, w))
1143 10        adj[v].append((u, w))
1144 11    return adj
1145 12
1146 13 def dijkstra(adj, src):
1147 14     from heapq import heappush, heappop
1148 15     INF = 10**18
1149 16     n = len(adj)
1150 17     dist = [INF]*n
1151 18     parent = [-1]*n
1152 19     dist[src] = 0
1153 20     heap = [(0, src)]
1154 21     while heap:
1155 22         d, u = heappop(heap)
1156 23         if d > dist[u]:
1157 24             continue
1158 25         for v, w in adj[u]:
1159 26             nd = d + w
1160 27             if nd < dist[v]:
1161 28                 dist[v] = nd
1162 29                 parent[v] = u
1163 30                 heappush(heap, (nd, v))
1164 31    return dist, parent
1165 32
1166 33 def reconstruct_path(parent, dest):
1167 34     path = []
1168 35     u = dest
1169 36     while u != -1:
1170 37         path.append(u+1)
1171 38         u = parent[u]
1172 39     return path[::-1]
1173 40
1174 41 def multi_source_bfs(neighbors, sources):
1175 42     from collections import deque
1176 43     n = len(neighbors)
1177 44     dist = [-1]*n
1178 45     dq = deque()
1179 46     for u in sources:
1180 47         if dist[u] == -1:
1181 48             dist[u] = 0
1182 49             dq.append(u)
1183 50     while dq:
1184 51         u = dq.popleft()
1185 52         for v in neighbors[u]:
1186 53             if dist[v] == -1:
1187 54                 dist[v] = dist[u] + 1
1188 55                 dq.append(v)
1189 56    return dist

```

Listing 6: Version 1, New Helpers

```

1188 1 ##### PROGRAM: node_16:cc_python_16 #####
1189 2
1190 3 from codebank import *
1191 4
1192 5 def main():
1193 6     import heapq
1194 7     n, m = read_ints()
1195 8     edges = [(u-1, v-1, w) for u, v, w in (read_ints() for _ in range(m))]
1196 9     adj = build_adj_undirected(n, edges)
1197 10    INF = 10**20
1198 11    dist = [INF]*n
1199 12    dist[0] = 0
1200 13    last_w = [0]*n
1201 14    heap = [(0, 0)]
1202 15    while heap:
1203 16        d, u = heapq.heappop(heap)
1204 17        if d > dist[u]:
1205 18            continue
1206 19        # record last edges
1207 20        for v, w in adj[u]:
1208 21            last_w[v] = w
1209 22        # expand two-edge moves
1210 23        for v, w1 in adj[u]:
1211 24            tw = last_w[v]
1212 25            for x, w2 in adj[v]:
1213 26                nd = d + (tw + w2)**2
1214 27                if nd < dist[x]:
1215 28                    dist[x] = nd
1216 29                    heapq.heappush(heap, (nd, x))
1217 30
1218 31    out = []
1219 32    for x in dist:
1220 33        out.append(str(x if x < INF else -1))
1221 34    print(" ".join(out))
1222 35
1223 36 if __name__ == "__main__":
1224 37     main()
1225 38
1226 39 ##### PROGRAM: node_17:cc_python_17 #####
1227 40
1228 41 from codebank import *
1229 42
1230 43 def main():
1231 44     n, m = read_ints()
1232 45     edges = [(u-1, v-1, w) for u, v, w in (read_ints() for _ in range(m))]
1233 46     adj = build_adj_undirected(n, edges)
1234 47     dist, parent = dijkstra(adj, 0)
1235 48     if dist[n-1] >= 10**18:
1236 49         print(-1)
1237 50     else:
1238 51         path = reconstruct_path(parent, n-1)
1239 52         print(*path)
1240 53
1241 54 if __name__ == "__main__":
1242 55     main()
1243 56
1244 57 ##### PROGRAM: node_19:cc_python_19 #####
1245 58
1246 59 from codebank import *
1247 60
1248 61 def main():
1249 62     n = int(input())
1250 63     a = read_ints()
1251 64     # build reversed graph: for each move i->j, add edge j->i
1252 65     neighbors = [[] for _ in range(n)]
1253 66     for i, val in enumerate(a):
1254 67         for j in (i - val, i + val):
1255 68             if 0 <= j < n:
1256 69                 neighbors[j].append(i)
1257 70     # BFS from all even and all odd positions separately
1258 71     even_sources = [i for i, val in enumerate(a) if val % 2 == 0]
1259 72     odd_sources = [i for i, val in enumerate(a) if val % 2 == 1]
1260 73     dist_even = multi_source_bfs(neighbors, even_sources)
1261 74     dist_odd = multi_source_bfs(neighbors, odd_sources)
1262 75     # for odd a[i], answer is dist to nearest even => dist_even; else dist_odd
1263 76     ans = [dist_even[i] if a[i] % 2 == 1 else dist_odd[i] for i in range(n)]
1264 77     print(*ans)
1265 78
1266 79 if __name__ == "__main__":
1267 80     main()

```

Listing 7: Version 1, Refactored Programs

```

1242 1 #
1243 2 # ==== NEW HELPER FUNCTIONS ====
1244 3 def read_ints():
1245 4     return list(map(int, input().split()))
1246 5
1247 6 def build_undirected_weighted_graph(n, m):
1248 7     from collections import defaultdict
1249 8     adj = defaultdict(list)
1250 9     for _ in range(m):
1251 10         u, v, w = read_ints()
1252 11         u -= 1; v -= 1
1253 12         adj[u].append((v, w))
1254 13         adj[v].append((u, w))
1255 14     return adj
1256 15
1257 16 def dijkstra(adj, src, n):
1258 17     import heapq
1259 18     INF = 10**18
1260 19     dist = [INF]*n
1261 20     parent = [-1]*n
1262 21     visited = [False]*n
1263 22     dist[src] = 0
1264 23     heap = [(0, src)]
1265 24     while heap:
1266 25         d, u = heapq.heappop(heap)
1267 26         if visited[u]:
1268 27             continue
1269 28         visited[u] = True
1270 29         for v, w in adj.get(u, ()):
1271 30             nd = d + w
1272 31             if nd < dist[v]:
1273 32                 dist[v] = nd
1274 33                 parent[v] = u
1275 34                 heapq.heappush(heap, (nd, v))
1276 35     return dist, parent
1277 36
1278 37 def reconstruct_path(parent, dest):
1279 38     path = []
1280 39     while dest != -1:
1281 40         path.append(dest+1)
1282 41         dest = parent[dest]
1283 42     return path[::-1]
1284 43
1285 44 def dijkstra_special(e, n, src):
1286 45     import heapq
1287 46     INF = 10**18
1288 47     d = [INF]*n
1289 48     d[src] = 0
1290 49     heap = [(0, src)]
1291 50     while heap:
1292 51         cd, v = heapq.heappop(heap)
1293 52         if cd > d[v]:
1294 53             continue
1295 54         td = {}
1296 55         for u, w in e.get(v, ()):
1297 56             td[u] = w
1298 57         for u, w1 in td.items():
1299 58             for x, w2 in e.get(u, ()):
1300 59                 cost = cd + (w1 + w2)**2
1301 60                 if cost < d[x]:
1302 61                     d[x] = cost
1303 62                     heapq.heappush(heap, (cost, x))
1304 63     return d
1305 64
1306 65 def compute_min_moves_opposite_parity(a):
1307 66     from collections import deque
1308 67     n = len(a)
1309 68     go = [[] for _ in range(n)]
1310 69     ans = [-1]*n
1311 70     q = deque()
1312 71     for i, val in enumerate(a):
1313 72         for j in (i - val, i + val):
1314 73             if 0 <= j < n:
1315 74                 if (a[j] % 2) != (val % 2):
1316 75                     ans[i] = 1
1317 76                     q.append(i)
1318 77                     break
1319 78             else:
1320 79                 go[j].append(i)
1321 80     while q:
1322 81         u = q.popleft()
1323 82         for v in go[u]:
1324 83             if ans[v] == -1:
1325 84                 ans[v] = ans[u] + 1
1326 85                 q.append(v)
1327 86     return ans

```

Listing 8: Version 2, New Helpers

```

1296 1 #
1297 2 # ##### PROGRAM: node_16:cc_python_16 #####
1298 3
1299 4 from codebank import *
1300 5
1301 6 def main():
1302 7     n, m = read_ints()
1303 8     e = {}
1304 9     for _ in range(m):
1305 10         u, v, w = read_ints()
1306 11         u -= 1; v -= 1
1307 12         e.setdefault(u, []).append((v, w))
1308 13         e.setdefault(v, []).append((u, w))
1309 14         d = dijkstra_special(e, n, 0)
1310 15         print(" ".join(str(-1 if x >= 10**18 else int(x)) for x in d))
1311 16
1312 17 if __name__ == "__main__":
1313 18     main()
1314 19
1315 20 # ##### PROGRAM: node_17:cc_python_17 #####
1316 21
1317 22 from codebank import *
1318 23
1319 24 def main():
1320 25     n, m = read_ints()
1321 26     adj = build_undirected_weighted_graph(n, m)
1322 27     dist, parent = dijkstra(adj, 0, n)
1323 28     if dist[n-1] >= 10**18:
1324 29         print(-1)
1325 30     else:
1326 31         path = reconstruct_path(parent, n-1)
1327 32         print(" ".join(map(str, path)))
1328 33
1329 34 if __name__ == "__main__":
1330 35     main()
1331 36
1332 37 # ##### PROGRAM: node_19:cc_python_19 #####
1333 38
1334 39 from codebank import *
1335 40
1336 41 def main():
1337 42     n = int(input())
1338 43     a = read_ints()
1339 44     ans = compute_min_moves_opposite_parity(a)
1340 45     print(" ".join(map(str, ans)))
1341 46
1342 47 if __name__ == "__main__":
1343 48     main()

```

Listing 9: Version 2, Refactored Programs

K OBFUSCATION EXAMPLE

We provide the full source code, both refactored and obfuscated, for the MDL and token comparison here.

Listing 10: Refactored code from modeling_llama.py from the refactored Transformers repository.

```

1336 from typing import Optional, Union
1337
1338 import torch
1339 from torch import nn
1340
1341 from ...cache_utils import Cache, DynamicCache
1342 from ...generation import GenerationMixin
1343 from ...masking_utils import create_causal_mask
1344 from ...modeling_layers import (
1345     GenericForQuestionAnswering,
1346     GenericForSequenceClassification,
1347     GenericForTokenClassification,
1348 )
1349 from ...modeling_outputs import (
1350     BaseModelOutputWithPast,
1351     CausalLMOutputWithPast,
1352 )
1353 from ...modeling_utils import PreTrainedModel
1354 from ...processing_utils import Unpack
1355 from ...utils import TransformersKwargs, auto_docstring, can_return_tuple, logging
1356 from ...utils.generic import check_model_inputs
1357 from .configuration_llama import LlamaConfig
1358
1359 from ..shared_library import (

```

```

1350         rotate_half,
1351         apply_rotary_pos_emb,
1352         repeat_kv,
1353         eager_attention_forward,
1354         RMSNorm,
1355         BaseMLP,
1356         BaseRotaryEmbedding,
1357         BaseAttention,
1358         BaseDecoderLayer,
1359     )
1360
1361     logger = logging.get_logger(__name__)
1362
1363     class LlamaRMSNorm(RMSNorm):
1364         pass
1365
1366     class LlamaRotaryEmbedding(BaseRotaryEmbedding):
1367         pass
1368
1369     class LlamaMLP(BaseMLP):
1370         def __init__(self, config):
1371             super().__init__(config, mlp_bias=config.mlp_bias)
1372
1373     class LlamaAttention(BaseAttention):
1374         def __init__(self, config: LlamaConfig, layer_idx: int):
1375             super().__init__(
1376                 config=config,
1377                 layer_idx=layer_idx,
1378                 attention_bias=config.attention_bias,
1379                 sliding_window=None
1380             )
1381
1382     class LlamaDecoderLayer(BaseDecoderLayer):
1383         def __init__(self, config: LlamaConfig, layer_idx: int):
1384             super().__init__(
1385                 config=config,
1386                 layer_idx=layer_idx,
1387                 norm_class=LlamaRMSNorm,
1388                 mlp_class=LlamaMLP,
1389                 attention_class=LlamaAttention
1390             )
1391
1392     @auto_docstring
1393     class LlamaPreTrainedModel(PreTrainedModel):
1394         config: LlamaConfig
1395         base_model_prefix = "model"
1396         supports_gradient_checkpointing = True
1397         _no_split_modules = ["LlamaDecoderLayer"]
1398         _skip_keys_device_placement = ["past_key_values"]
1399         _supports_flash_attn = True
1400         _supports_sdpa = True
1401         _supports_flex_attn = True
1402
1403         _can_compile_fullgraph = True
1404         _supports_attention_backend = True
1405         _can_record_outputs = {
1406             "hidden_states": LlamaDecoderLayer,
1407             "attentions": LlamaAttention,
1408         }
1409
1410     @auto_docstring
1411     class LlamaModel(LlamaPreTrainedModel):
1412         def __init__(self, config: LlamaConfig):
1413             super().__init__(config)
1414             self.padding_idx = config.pad_token_id
1415             self.vocab_size = config.vocab_size
1416
1417             self.embed_tokens = nn.Embedding(config.vocab_size, config.hidden_size, self.padding_idx)
1418             self.layers = nn.ModuleList(
1419                 [LlamaDecoderLayer(config, layer_idx) for layer_idx in range(config.num_hidden_layers)]
1420             )
1421             self.norm = LlamaRMSNorm(config.hidden_size, eps=config.rms_norm_eps)
1422             self.rotary_emb = LlamaRotaryEmbedding(config=config)

```

```

1404         self.gradient_checkpointing = False
1405
1406         self.post_init()
1407
1408     @check_model_inputs
1409     @auto_docstring
1410     def forward(
1411         self,
1412         input_ids: Optional[torch.LongTensor] = None,
1413         attention_mask: Optional[torch.Tensor] = None,
1414         position_ids: Optional[torch.LongTensor] = None,
1415         past_key_values: Optional[Cache] = None,
1416         inputs_embeds: Optional[torch.FloatTensor] = None,
1417         cache_position: Optional[torch.LongTensor] = None,
1418         use_cache: Optional[bool] = None,
1419         **kwargs: Unpack[TransformersKwargs],
1420     ) -> BaseModelOutputWithPast:
1421         if (input_ids is None) ^ (inputs_embeds is not None):
1422             raise ValueError("You must specify exactly one of input_ids or inputs_embeds")
1423
1424         if inputs_embeds is None:
1425             inputs_embeds: torch.Tensor = self.embed_tokens(input_ids)
1426
1427         if use_cache and past_key_values is None:
1428             past_key_values = DynamicCache(config=self.config)
1429
1430         if cache_position is None:
1431             past_seen_tokens = past_key_values.get_seq_length() if past_key_values is not None else 0
1432             cache_position: torch.Tensor = torch.arange(
1433                 past_seen_tokens, past_seen_tokens + inputs_embeds.shape[1], device=inputs_embeds.device
1434             )
1435
1436         if position_ids is None:
1437             position_ids = cache_position.unsqueeze(0)
1438
1439         causal_mask = create_causal_mask(
1440             config=self.config,
1441             input_embeddings=inputs_embeds,
1442             attention_mask=attention_mask,
1443             cache_position=cache_position,
1444             past_key_values=past_key_values,
1445             position_ids=position_ids,
1446         )
1447
1448         hidden_states = inputs_embeds
1449         position_embeddings = self.rotary_emb(hidden_states, position_ids)
1450
1451         for decoder_layer in self.layers[: self.config.num_hidden_layers]:
1452             hidden_states = decoder_layer(
1453                 hidden_states,
1454                 attention_mask=causal_mask,
1455                 position_ids=position_ids,
1456                 past_key_values=past_key_values,
1457                 cache_position=cache_position,
1458                 position_embeddings=position_embeddings,
1459                 **kwargs,
1460             )
1461
1462         hidden_states = self.norm(hidden_states)
1463         return BaseModelOutputWithPast(
1464             last_hidden_state=hidden_states,
1465             past_key_values=past_key_values,
1466         )
1467
1468     @auto_docstring
1469     class LlamaForCausalLM(LlamaPreTrainedModel, GenerationMixin):
1470         _tied_weights_keys = ["lm_head.weight"]
1471         _tp_plan = {"lm_head": "colwise_rep"}
1472         _pp_plan = {"lm_head": ([ "hidden_states" ], [ "logits" ])}
1473
1474         def __init__(self, config):
1475             super().__init__(config)
1476             self.model = LlamaModel(config)
1477             self.vocab_size = config.vocab_size
1478             self.lm_head = nn.Linear(config.hidden_size, config.vocab_size, bias=False)
1479
1480             self.post_init()
1481
1482         def set_decoder(self, decoder):
1483             self.model = decoder

```

```

1458
1459     def get_decoder(self):
1460         return self.model
1461
1462     @can_return_tuple
1463     @auto_docstring
1464     def forward(
1465         self,
1466         input_ids: Optional[torch.LongTensor] = None,
1467         attention_mask: Optional[torch.Tensor] = None,
1468         position_ids: Optional[torch.LongTensor] = None,
1469         past_key_values: Optional[Cache] = None,
1470         inputs_embeds: Optional[torch.FloatTensor] = None,
1471         labels: Optional[torch.LongTensor] = None,
1472         use_cache: Optional[bool] = None,
1473         cache_position: Optional[torch.LongTensor] = None,
1474         logits_to_keep: Union[int, torch.Tensor] = 0,
1475         **kwargs: Unpack[TransformersKwargs],
1476     ) -> CausalLMOutputWithPast:
1477         """
1478         Example:
1479
1480         ```python
1481         >>> from transformers import AutoTokenizer, LlamaForCausalLM
1482
1483         >>> model = LlamaForCausalLM.from_pretrained("meta-llama/Llama-2-7b-hf")
1484         >>> tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf")
1485
1486         >>> prompt = "Hey, are you conscious? Can you talk to me?"
1487         >>> inputs = tokenizer(prompt, return_tensors="pt")
1488
1489         >>> # Generate
1490         >>> generate_ids = model.generate(inputs.input_ids, max_length=30)
1491         >>> tokenizer.batch_decode(generate_ids, skip_special_tokens=True, clean_up_tokenization_spaces=False)[0]
1492         "Hey, are you conscious? Can you talk to me?\nI'm not conscious, but I can talk to you."
1493         """
1494         outputs: BaseModelOutputWithPast = self.model(
1495             input_ids=input_ids,
1496             attention_mask=attention_mask,
1497             position_ids=position_ids,
1498             past_key_values=past_key_values,
1499             inputs_embeds=inputs_embeds,
1500             use_cache=use_cache,
1501             cache_position=cache_position,
1502             **kwargs,
1503         )
1504
1505         hidden_states = outputs.last_hidden_state
1506         slice_indices = slice(-logits_to_keep, None) if isinstance(logits_to_keep, int) else logits_to_keep
1507         logits = self.lm_head(hidden_states[:, slice_indices, :])
1508
1509         loss = None
1510         if labels is not None:
1511             loss = self.loss_function(logits=logits, labels=labels, vocab_size=self.config.vocab_size, **kwargs)
1512
1513         return CausalLMOutputWithPast(
1514             loss=loss,
1515             logits=logits,
1516             past_key_values=outputs.past_key_values,
1517             hidden_states=outputs.hidden_states,
1518             attentions=outputs.attentions,
1519         )
1520
1521 class LlamaForSequenceClassification(GenericForSequenceClassification, LlamaPreTrainedModel): ...
1522
1523 class LlamaForQuestionAnswering(GenericForQuestionAnswering, LlamaPreTrainedModel):
1524     base_model_prefix = "transformer"
1525
1526 class LlamaForTokenClassification(GenericForTokenClassification, LlamaPreTrainedModel): ...
1527
1528 __all__ = [
1529     "LlamaForCausalLM",
1530     "LlamaModel",
1531     "LlamaPreTrainedModel",
1532     "LlamaForSequenceClassification",
1533     "LlamaForQuestionAnswering",
1534     "LlamaForTokenClassification",

```

]

Listing 11: Obfuscated code from modeling_llama.py from the refactored Transformers repository.

```

from typing import Optional as O, Union as U
import torch as T
from torch import nn as N
from ...cache_utils import Cache as C, DynamicCache as D
from ...generation import GenerationMixin as G
from ...masking_utils import create_causal_mask as m
from ...modeling_layers import GenericForQuestionAnswering as Q, GenericForSequenceClassification as S,
    GenericForTokenClassification as K
from ...modeling_outputs import BaseModelOutputWithPast as B, CausalLMOutputWithPast as L
from ...modeling_utils import PreTrainedModel as P
from ...processing_utils import Unpack as X
from ...utils import TransformersKwargs as W, auto_docstring as a, can_return_tuple as r, logging
from ...utils.generic import check_model_inputs as c
from .configuration_llama import LlamaConfig as F
from ..shared_library import rotate_half, apply_rotary_pos_emb, repeat_kv, eager_attention_forward, RMSNorm as R,
    BaseMLP as M, BaseRotaryEmbedding as E, BaseAttention as A, BaseDecoderLayer as Y
l=logging.get_logger(__name__)
class Z(R):pass
class I(E):pass
class J(M):
    def __init__(s,g):super().__init__(g,mlp_bias=g.mlp_bias)
class H(A):
    def __init__(s,g:F,i:int):super().__init__(config=g,layer_idx=i,attention_bias=g.attention_bias,sliding_window=
        None)
class V(Y):
    def __init__(s,g:F,i:int):super().__init__(config=g,layer_idx=i,norm_class=Z,mlp_class=J,attention_class=H)
@a
class o(P):
    config:F
    base_model_prefix="model"
    supports_gradient_checkpointing=True
    _no_split_modules=["LlamaDecoderLayer"]
    _skip_keys_device_placement="past_key_values"
    _supports_flash_attn=True
    _supports_sdpa=True
    _supports_flex_attn=True
    _can_compile_fullgraph=True
    _supports_attention_backend=True
    _can_record_outputs={"hidden_states":V,"attentions":H}
@a
class u(o):
    def __init__(s,g:F):
        super().__init__(g)
        s.padding_idx=g.pad_token_id
        s.vocab_size=g.vocab_size
        s.embed_tokens=N.Embedding(g.vocab_size,g.hidden_size,s.padding_idx)
        s.layers=N.ModuleList([V(g,i)for i in range(g.num_hidden_layers)])
        s.norm=Z(g.hidden_size,eps=g.rms_norm_eps)
        s.rotary_emb=I(config=g)
        s.gradient_checkpointing=False
        s.post_init()
    @c
    @a
    def forward(s,input_ids:O[T.LongTensor]=None,attention_mask:O[T.Tensor]=None,position_ids:O[T.LongTensor]=None,
        past_key_values:O[C]=None,inputs_embeds:O[T.FloatTensor]=None,cache_position:O[T.LongTensor]=None,use_cache:
        O[bool]=None,**k:X[W])>B:
        if(input_ids is None)^(inputs_embeds is not None):raise ValueError("You must specify exactly one of input_ids
            or inputs_embeds")
        if inputs_embeds is None:inputs_embeds:T.Tensor=s.embed_tokens(input_ids)
        if use_cache and past_key_values is None:past_key_values=D(config=s.config)
        if cache_position is None:
            p=past_key_values.get_seq_length()if past_key_values is not None else 0
            cache_position:T.Tensor=T.arange(p,p+inputs_embeds.shape[1],device=inputs_embeds.device)
        if position_ids is None:position_ids=cache_position.unsqueeze(0)
        f=m(config=s.config,input_embeds=inputs_embeds,attention_mask=attention_mask,cache_position=cache_position,
            past_key_values=past_key_values,position_ids=position_ids)
        h=inputs_embeds
        e=s.rotary_emb(h,position_ids)
        for d in s.layers[:s.config.num_hidden_layers]:h=d(h,attention_mask=f,position_ids=position_ids,past_key_values
            =past_key_values,cache_position=cache_position,position_embeddings=e,**k)
        h=s.norm(h)
        return B(last_hidden_state=h,past_key_values=past_key_values)
@a
class t(o,G):
    _tied_weights_keys=["lm_head.weight"]
    _tp_plan={"lm_head":"colwise_rep"}

```

```

1566 _pp_plan={"lm_head":(["hidden_states"],["logits"])}
1567 def __init__(s,g):
1568     super().__init__(g)
1569     s.model=u(g)
1569     s.vocab_size=g.vocab_size
1570     s.lm_head=N.Linear(g.hidden_size,g.vocab_size,bias=False)
1570     s.post_init()
1571     def set_decoder(s,d):s.model=d
1572     def get_decoder(s):return s.model
1573     @r
1573     @a
1574     def forward(s,input_ids:0[T.LongTensor]=None,attention_mask:0[T.Tensor]=None,position_ids:0[T.LongTensor]=None,
1575         past_key_values:0[C]=None,inputs_embeds:0[T.FloatTensor]=None,labels:0[T.LongTensor]=None,use_cache:0[bool
1575         ]=None,cache_position:0[T.LongTensor]=None,logits_to_keep:U[int,T.Tensor]=0,**k:X[W])->L:
1576     o:B=s.model(input_ids=input_ids,attention_mask=attention_mask,position_ids=position_ids,past_key_values=
1576         past_key_values,inputs_embeds=inputs_embeds,use_cache=use_cache,cache_position=cache_position,**k)
1577     h=o.last_hidden_state
1578     i=slice(-logits_to_keep,None)if isinstance(logits_to_keep,int)else logits_to_keep
1578     g=s.lm_head(h[:,i,:])
1579     n=None
1580     if labels is not None:n=s.loss_function(logits=g,labels=labels,vocab_size=s.config.vocab_size,**k)
1580     return L(loss=n,logits=g,past_key_values=o.past_key_values,hidden_states=o.hidden_states,attentions=o.
1581         attentions)
1582 class b(S,o):...
1582 class x(Q,o):base_model_prefix="transformer"
1583 class y(K,o):...
1584 __all__=["t","u","o","b","x","y"]
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

```