# Discovering and Detecting Tax Avoidance Using Natural Language Processing and Coevolutionary Algorithms

by

Joy Sera Bhattacharya

S.B. Computer Science, Economics, and Data Science, MIT (2024)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTER SCIENCE, ECONOMICS, AND DATA SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

© 2025 Joy Sera Bhattacharya. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Joy Sera Bhattacharya
Department of Electrical Engineering and Computer Science
May 2025

Certified by: Una-May O'Reilly
Principal Research Scientist, Thesis Supervisor

Certified by: Erik Hemberg
Research Scientist, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

# Discovering and Detecting Tax Avoidance Using Natural Language Processing and Coevolutionary Algorithms

by

Joy Sera Bhattacharya

Submitted to the Department of Electrical Engineering and Computer Science
on May 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTER SCIENCE, ECONOMICS, AND DATA SCIENCE

## ABSTRACT

The Internal Revenue Service estimates that unpaid taxes cost the United States up to $1 trillion each year, much of it through tax avoidance schemes such as the Installment Bogus Optional Basis strategy (iBOB). This thesis focuses on discovering and detecting iBOB schemes within a tax network by using a coevolutionary framework powered by two large language model (LLM) agents: a tax planner, which generates transactions to reduce tax and suspicion, and an auditor, which generates patterns to detect tax avoidance. This thesis shows that as the tax planner and auditor evolve against each other, the planner consistently uncovers iBOB transactions and even discovers more effective variants than those previously known. In turn, the auditor learns to generate increasingly targeted detection patterns that adapt to the planner's changing strategies. Their interaction leads to oscillating behavior between avoidance and enforcement, providing a new LLM-based framework for studying the dynamics of tax compliance.

Thesis supervisor: Una-May O'Reilly

Title: Principal Research Scientist

Thesis supervisor: Erik Hemberg

Title: Research Scientist

# Acknowledgments

First and foremost, I'm incredibly grateful to my amazing advisors, Una-May O'Reilly and Erik Hemberg. Their support was constant, their mentorship invaluable, and their humor a much needed source of sanity.

To everyone at ALFA, thank you for being so brilliant and supportive. Your camaraderie made every day in the lab something to look forward to.

A huge thank you to my collaborators in Colombia; your dedication, generosity, and passion inspired me more than you realize. You pushed me to work harder, aim higher, and pick up some Spanish slang along the way.

To the community of friends, cheerleaders, and, of course, the bunnies outside Stata at night, you were the reason I made it through. A special thank you to my friends for always being in my corner, for building a home away from home, for indulging my antics, and for never making me feel crazy, even when I was definitely testing your patience.

And finally, to my family, thank you for your constant encouragement and for always finding ways to make me laugh. Your steadfast belief in me, especially when I struggled to find it in myself, made all the difference. I am indescribably lucky to be related to you all.

# Contents

*List of Figures* 11

*List of Tables* 13

**1 Introduction** **15**

1.1 Background . . . . . 16

1.2 Research Questions . . . . . 16

1.3 Approach . . . . . 17

1.4 Contributions . . . . . 18

1.5 Outline . . . . . 20

**2 Background and Related Work** **21**

2.1 Tax Background . . . . . 21

2.1.1 iBOB Scheme . . . . . 25

2.2 Preliminary Study . . . . . 27

2.3 Related Work . . . . . 31

**3 Methodology** **35**

3.1 Environment . . . . . 36

3.2 Transaction and Pattern Generation . . . . . 39

3.3 Transaction and Pattern Search . . . . . 42

**4 Experiments** **47**

4.1 LLM-Tax Planner . . . 48

4.1.1 T1: Transaction Generation . . . 51

4.1.2 T2: Transaction Variation . . . 53

4.1.3 T3: Additional Experiments . . . 55

4.2 LLM-Auditor . . . 55

4.2.1 A1: Pattern Generation . . . 58

4.2.2 A2: Pattern Variation . . . 59

4.2.3 A3: Additional Experiments . . . 60

4.3 Coevolution Experiments . . . 61

4.3.1 C1: Fitness Function Comparison . . . 64

4.3.2 C2: Coevolution vs. Benchmarks . . . 72

4.3.3 C3: Generations and Pattern Richness . . . 75

**5 Conclusion** **79**

5.1 Summary . . . 79

5.2 Limitations . . . 82

5.3 Future Work . . . 83

*References* 85

**A Appendix A: Algorithms** **89**

A.1 Generation Algorithms . . . 89

A.2 Search Algorithms . . . 91

**B Appendix B: Prompts** **93**

B.1 Prompts Used . . . . . 93

B.1.1 LLM-Tax Planner . . . . . 93

B.1.2 LLM-Auditor . . . . . 97

B.2 Prompts Tested During Sensitivity Experiments . . . . . 99

B.2.1 LLM-Tax Planner . . . . . 99

B.2.2 LLM-Auditor . . . . . 112

**C Appendix C: Additional Experiments** **117**

C.1 T*: LLM-Tax Planner Experiments . . . . . 117

C.1.1 Mutation-Prompt Sensitivity Experiment . . . . . 117

C.2 A*: LLM-Auditor Experiments . . . . . 118

C.2.1 Number of Patterns Experiment . . . . . 118

C.2.2 Co-occurring Patterns Experiment . . . . . 119

C.3 C*: Coevolution Experiments . . . . . 120

C.3.1 Examples of Fitness Calculations . . . . . 120

C.3.2 Fitness Function Comparison Experiment . . . . . 121

C.3.3 Coevolution vs. Benchmarks Experiment . . . . . 123

# List of Figures


1.1 Overview of Coevolutionary Setup . . . . . 18

2.1 Tax Liability Calculation Flow Chart . . . . . 23

2.2 Tax Network with iBOB Scheme . . . . . 26

2.3 GPT-4o's iBOB Discovery . . . . . 28

2.4 GPT-4o iBOB Discovery and Ranking . . . . . 29

2.5 GPT-4o Contradicting Legal Assessment of Scheme . . . . . 30

2.6 GPT-4o Misclassification of iBOB as Legal . . . . . 31

3.1 Overview of Environmental Pipeline . . . . . 37

3.2 Graph G: Initial Tax Network . . . . . 38

3.3 Generating Transaction Sequences . . . . . 40

3.4 Generating Patterns . . . . . 42

3.5 Transaction Evolution . . . . . 44

3.6 Pattern Evolution . . . . . 45

4.1 Formal Grammar Used in Select Generation Prompts . . . . . 52

4.2 Four Patterns: Avg Number of iBOB Transactions . . . . . 77

4.3 Four Patterns: Avg Fitness . . . . . 77

4.4 Ten Patterns: Avg Number of iBOB Transactions . . . . . 77

4.5 Ten Patterns: Avg Fitness . . . . . 77

C.1 Fitness 1 Results . . . . . 121

C.2 Fitness 2 Results . . . . . 121

C.3 Fitness 3 Results . . . . . 121

C.4 Fitness 4 Results . . . . . 122

C.5 Fitness 5 Results . . . . . 122

C.6 Audit Evolution with Static Tax Planner Results . . . . . 123

C.7 Transaction Evolution with Static Auditor Results . . . . . 123

# List of Tables


2.1 Summary of Preliminary Results with GPT-4o . . . . . 27

2.2 Overview of Related Work by Method and Focus . . . . . 34

4.1 LLM-Tax Planner Experiments . . . . . 49

4.2 Hyperparameters Used in LLM-Tax Planner Experiments . . . . . 50

4.3 Evaluation Metrics Used in LLM-Tax Planner Experiments . . . . . 51

4.4 LLM-Tax Planner, Transaction Generation Experiment . . . . . 53

4.5 LLM-Tax Planner, Transaction Variation Experiment . . . . . 54

4.6 LLM-Auditor Experiments . . . . . 56

4.7 Hyperparameters Used in LLM-Auditor Experiments . . . . . 57

4.8 Evaluation Metrics Used in LLM-Auditor Experiments . . . . . 58

4.9 LLM-Auditor, Pattern Generation Experiment . . . . . 59

4.10 LLM-Auditor, Pattern Variation Experiment . . . . . 60

4.11 Summary of Coevolution Experiments . . . . . 62

4.12 Hyperparameters Used in Coevolution Experiments . . . . . 63

4.13 Evaluation Metrics Used in Coevolution Experiment . . . . . 64

4.14 Coevolution, Coevolution Fitness Function Comparison Experiment . . . . . 71

4.15 Coevolution, Coevolution vs. Benchmarks Experiment . . . . . 74

4.16 Coevolution, Generations and Pattern Richness Experiment . . . . . 76

C.1 LLM-Tax Planner, Mutation Prompt Sensitivity Experiment . . . . . 118

C.2 LLM-Auditor, Number of Patterns Experiment . . . . . 119

C.3 LLM-Auditor, Co-occurring Patterns Experiment . . . . . 120

C.4 Example of Fitness Calculations on Transaction Sequences . . . . . 120

C.5 Example of Fitness Calculations on Patterns . . . . . 120

# 1

# Introduction

The Internal Revenue Service estimates that unpaid taxes cost the United States as much as $1 trillion each year (Rappeport 2021). A considerable share of this loss stems from avoidance schemes that exploit gaps in the tax code, like the Installment Bogus Optional Basis (iBOB) strategy. **Discovering and Detecting Tax Avoidance using Natural Language Processing and Coevolutionary Algorithms** addresses this problem with an agent based framework that integrates large language models and evolutionary search to simulate the dynamic between taxpayers and auditors. The framework uses two agents: an LLM-Tax Planner, which generates and adjusts transaction sequences to lower liabilities while limiting audit exposure, and an LLM-Auditor, which iteratively refines audit patterns to flag questionable transactions. At every generation, a fitness function penalizes the planner for higher tax and greater audit risk and rewards the auditor for detecting suspicious transactions, all of which drives a coevolutionary loop that mirrors the dynamics of real world enforcement.

Experimental results show that structured prompts, combined with search operators that mutate transactions, enable the planner to satisfy all tax network constraints while repeatedly discovering iBOB, including novel variants with lower tax burdens than previously expected. At the same time, the auditor evolves interpretable patterns that successfully detect these schemes and improve in performance over successive iterations. When tax penalties are incorporated into the fitness functions of both agents, their behaviors exhibit oscillation, and coevolution drives each to achieve higher fitness than any benchmark involving static or single-agent evolution. By linking a rule-based tax calculator with iterative adversarial training, this work bridges the gap between traditional hand-coded simulations and recent single-pass LLM audits, offering a platform for analyzing avoidance behavior, developing audit strategies, and informing tax policy.

## 1.1 Background

Previous studies on tax planning and evolutionary modeling illustrate how taxpayers and auditors continuously adjust their strategies in response to one another. For instance, agent-based models and genetic algorithms have shown how avoidance techniques evolve (Bloomquist 2011; Hemberg et al. 2016; Scott et al. 2024; Rosen 2015). In parallel, researchers in natural language processing (NLP) have attempted to convert tax statutes into machine-readable formats, encountering numerous challenges along the way (Pertierra et al. 2017). Modern large language models show considerable potential, yet they frequently misread statutory language and, at times, hallucinate legal provisions (Holzenberger, Blair-Stanek, and Durme 2020; Blair-Stanek, Holzenberger, and Durme 2023; Blair-Stanek, Holzenberger, and Durme 2024; Dahl et al. 2024; Zhong et al. 2020). To curb these errors, researchers have tested role-based prompting techniques and cycles of iterative refinement (S. Jung and J. Jung 2025; Lee et al. 2025; Liventsev et al. 2023). Together, these approaches highlight the need for adaptive computational methods to model tax avoidance.

## 1.2 Research Questions

This research focuses on answering several key questions about the use of LLMs in discovering and auditing avoidance schemes within tax networks. We group our questions into three main areas: generating and evolving transactions, generating and evolving audit patterns, and coevolution.

**Generating and Evolving Transactions**

- Can an LLM generate transaction sequences that follow the rules of a tax network?
- Can an LLM improve transaction sequences through evolution by using variations such as mutation, addition, and deletion?
- How sensitive is an LLM-Tax Planner's performance to prompt design, particularly when varying prompt structure and content?

**Generating and Evolving Audit Patterns**

- To what extent can an LLM detect and identify patterns in transaction data?

- Can an LLM improve audit patterns through evolution to better recognize suspicious transactions?
- How sensitive is an LLM-Auditor's performance to prompt design, particularly when varying prompt structure and content?

**Coevolution**

- How does the choice of a fitness function affect the convergence, oscillation, or diversity of transactions and patterns identified through coevolution?
- How does coevolution compare to benchmarks such as standalone pattern or transaction generation, sole LLM-Tax Planner evolution, or sole LLM-Auditor evolution?
- How does increasing the number of generations and audit patterns affect the long-run behavior of coevolution?

## 1.3 Approach

We first establish a structured environment of entities, assets, and partnerships. Within that setting, we introduce two coevolving LLM agents: a tax planner that proposes tax-minimizing transaction sequences and an auditor that flags suspicious patterns.

The tax planner generates complete sequences of transactions. The auditor produces generalizable patterns that signal suspicion. A rule-based tax calculator evaluates every sequence, and the auditor tags each one with any patterns it detects.

A fitness function drives the evolution of our agents. The tax planner tries to minimize tax liability while avoiding audit suspicion. In turn, the auditor tries to find audit patterns that are frequent and suspicious.

Evolution of agents happens in alternation. The tax planner first mutates its transaction sequences to lower tax and avoid detection. The auditor then mutates its patterns to capture the most common suspicious features in the new transaction sequences. This back-and-forth shows how evasion tactics and detection methods adapt together over time.

Figure 1.1 illustrates the loop. The environment stores the tax network, actions, and outcomes while the tax planner proposes transaction sequences and the auditor extracts patterns. Their

outputs feed back into the environment, which re-evaluates everything for the next round of evolution.

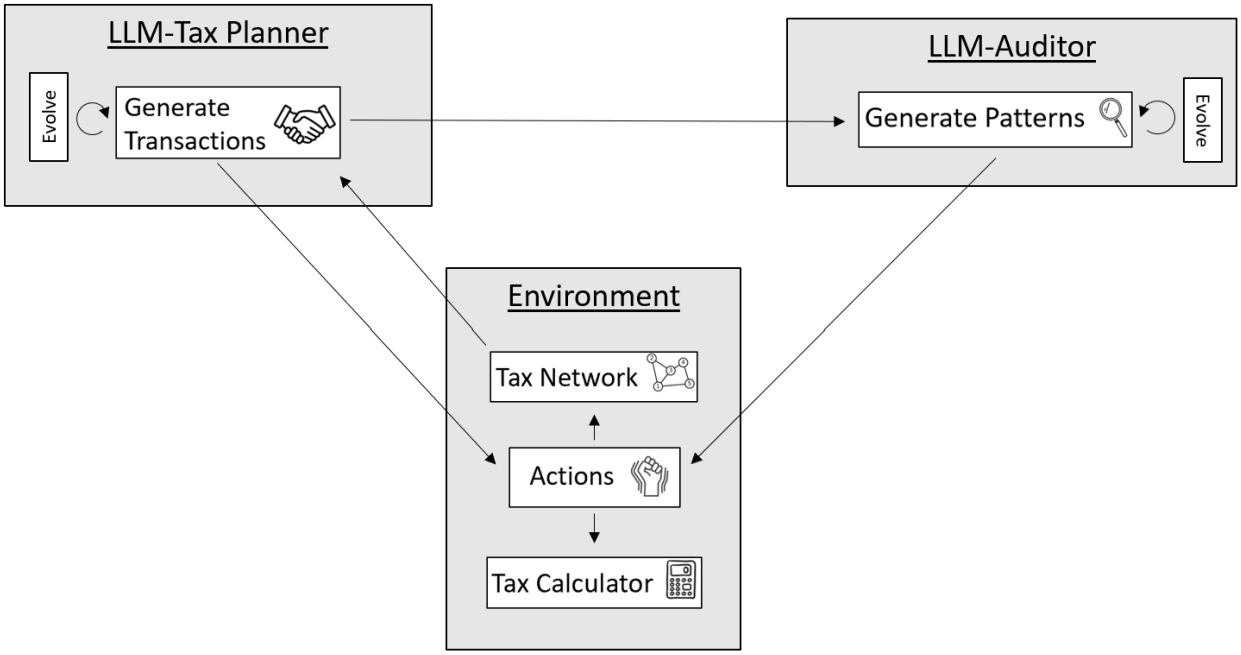


Figure 1.1: Overview of Coevolutionary Setup

**Notes:** A tax planner proposes tax-saving transaction sequences, an auditor extracts patterns that flag suspicious activity, and both agents iteratively adapt to each other within a shared environment that maintains the tax network, applies actions, and calculates liabilities.

## 1.4 Contributions

**Coevolutionary Framework** We present a coevolutionary search framework in which two large language model agents adapt to one another through successive cycles of mutation, evaluation, and selection. The LLM-Tax Planner evolves transaction sequences that minimize tax liabilities, whereas the LLM Auditor searches for patterns indicative of avoidance. Their sustained interaction reproduces the adversarial relationship found in tax enforcement. When applied to the iBOB avoidance scheme, the planner consistently discovers iBOB transactions and discovers even better iBOB variants, and the auditor learns to detect them, all under the constraints of a formal tax calculator. The framework therefore supports both the identification of known schemes and the discovery of novel variants.

Further, this work closes a gap between hand-coded agent models of tax planning (Hemberg

et al. 2016; Scott et al. 2024) and recent one-shot LLM audits (Fratrič, Holzenberger, and Amariles 2025). Whereas prior studies treat LLM reasoning as a single pass and risk statutory hallucinations (Blair-Stanek, Holzenberger, and Durme 2023; Dahl et al. 2024), we pair a manual tax calculator with a coevolutionary schedule that mirrors iterative refinement (Lee et al. 2025; Liventsev et al. 2023). Results show that coevolution creates an arms race between planners and auditors, giving researchers a way to study tax avoidance, adapt audit strategies, and design more effective tax policy. Below, we summarize our main contributions.

**LLM-Tax Planner**

- Structured prompts lower the number of semantic errors in generated transactions, but they also reduce the diversity of transactions.
- Mutation-only evolution most effectively discovers iBOB sequences, although it introduces a higher error rate.
- Detailed mutation prompts further improve evolutionary performance by increasing fitness and decreasing errors.

**LLM-Auditor**

- Minimal prompts yield the best results for the initial generation of audit patterns, whereas verbose prompts improve diversity and consistency during mutation. Optimal prompt type therefore depends on the nature of the task.
- Using four audit patterns to label ten transactions strikes an appropriate balance between coverage and redundancy, and allowing pattern co-occurrence enhances overall audit effectiveness.

**Coevolution of Tax Planner and Auditor**

- Under coevolutionary pressure, the planner discovers and the auditor detects iBOB strategies.
- Fitness functions that impose tax penalties create more realistic planner and auditor dynamics and avoid premature convergence on iBOB.

- The search identifies a previously unexpected iBOB variant, revealing weaknesses in the tax calculator that the planner can exploit and highlighting an LLM's potential to challenge expert knowledge.
- Across experimental settings, coevolution surpasses fixed agent baselines, producing more frequent iBOB discovery, higher agent fitness, and oscillatory dynamics.
- Expanding the auditor's pattern capacity delays iBOB's emergence and dominance.

## 1.5 Outline

The remainder of this thesis is organized as follows. Chapter 2 reviews the necessary tax concepts, introduces the iBOB avoidance scheme, summarizes preliminary findings, and positions this thesis within related work. Chapter 3 then details the tax network, LLM-agents, and coevolution that underpin our framework. Chapter 4 presents our empirical findings. Next, Chapter 5 summarizes our results, limitations, contributions, and outlines areas for future research. Finally, Appendix A presents all pseudocode, Appendix B provides all prompts, and Appendix C details further experiments and results.

# 2

# Background and Related Work

This section begins by explaining three central tax principles: fair market value, basis, and §754 elections (e.g. see Sec. 2.1). Together they form the foundation of partnership taxation and make the iBOB scheme possible, a strategy that removes taxable gain through basis manipulation while adding no genuine economic value (e.g. see Sec. 2.1.1). Our analysis shows that GPT-4o can uncover, formalize, and rank tactics of this kind. However, it does poorly when required to judge their legality or perform exact tax computations (e.g. see Sec. 2.2). To then justify the proposed coevolutionary planner–auditor framework, we examine prior work in tax planning, legal NLP, and evolutionary optimization and justify our contributions to this literature (e.g. see Sec. 2.3).

## 2.1 Tax Background

To understand the iBOB tax avoidance scheme, we first define tax concepts like fair market value (FMV), inside and outside basis, and optional basis adjustments (§754 elections), all covered by Subchapter K of the Internal Revenue Code (Legal Information Institute, Cornell Law School 2025). These concepts explain how tax liabilities are calculated in partnership transactions.

**Fair Market Value** Fair market value (FMV) refers to the selling price of an asset in an open and competitive market. FMV allows us to determine the taxable gain or loss when an asset is sold or transferred.

**Inside and Outside Basis** Basis represents a taxpayer's investment in an asset. Generally, basis is the original cost of the asset, adjusted over time for improvements, depreciation,

income, or distributions.

Basis is tracked at two levels in a partnership:

- **Inside basis (IB)**: the partnership's basis in its own assets.
- **Outside basis (OB)**: each partner's basis in their ownership interest.

When an asset is contributed to a partnership, the inside basis of that asset matches the contributing partner's basis in the asset. The partner's outside basis initially matches this amount. Over time, transactions such as income allocations, losses, and partnership sales can cause inside and outside bases to differ.

**Tax Calculation** Calculating the tax liability for a transaction depends on the type of asset being sold and the type of entity selling that asset. As Figure 2.1 shows below, loans, such as annuities or promissory notes, and cash are not taxed. On the other hand, if an individual tax payer is selling a material asset, their tax liability will be the difference in the asset's FMV and basis. If a partnership is selling a material asset, then each partner will have a tax liability of the difference between their share in the FMV (FMV*share) and their inside basis.

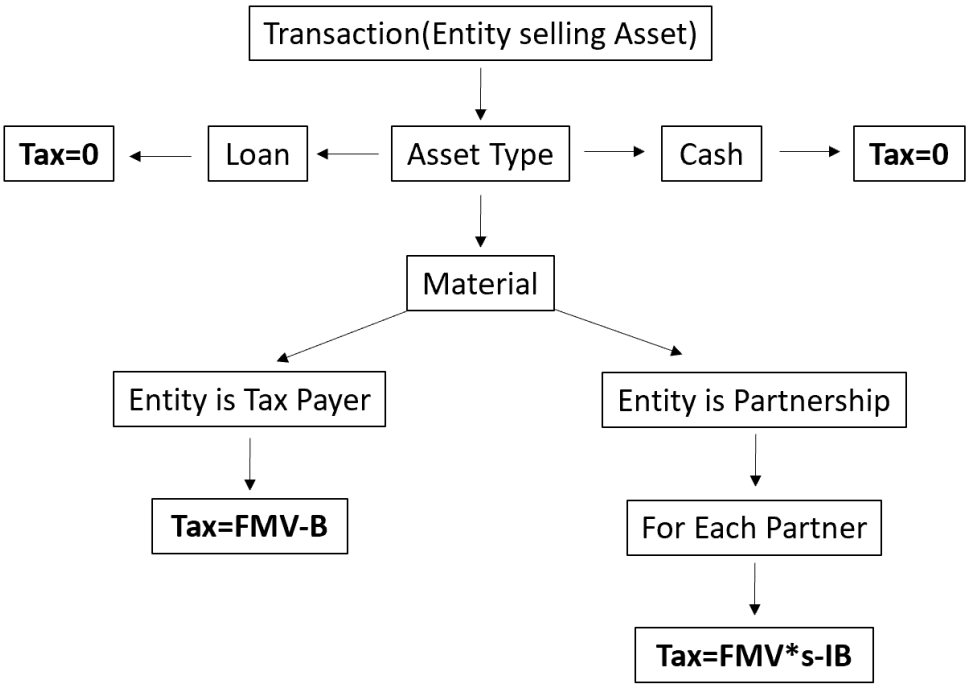


Figure 2.1: Tax Liability Calculation Flow Chart

**Notes:** Tax liability relies on two inputs: asset type and seller. Loans or cash aren't taxed, but a material asset is taxed on the difference between FMV and basis (B) for an individual, and each partner's share of that difference (FMV * s − inside basis) for a partnership.

**Example: Partnership Tax Calculation** For example, imagine that Jack and Jill form a 50/50 partnership. Jill contributes a house with a FMV of $200 and a basis of $120. Jack contributes equipment with a FMV of $200 and a basis of $60.

With these contributions:

- The partnership's inside basis is $120 for the house and $60 for the equipment.
- Jill's outside basis is $120.
- Jack's outside basis is $60.

Later, the partnership sells the house for $200. This results in:

- Sale price: $200
- Tax liability: $80 (FMV of $200 minus inside basis of $120)

With their equal ownership, Jack and Jill each have a tax liability of $40 from this gain.

**Sales and Transfers of Partnership Interests** When a partner sells their partnership interest to a new buyer, this buyer's outside basis in the partnership will equal the amount they paid for the interest. However, the buyer's inside basis in the partnership does not automatically change to match their outside basis. This can cause tax mismatches. For example, the new partner could inherit built-in gains or losses due to the difference in inside and outside basis.

**Example of Sale of Partnership Interest** Building on our earlier example, suppose Jill sells her partnership interest to Jerry for $200. Jerry's outside basis is now $200, but the inside basis of the partnership's assets remains unchanged at $120 for the house and $60 for the equipment.

Now, after the transfer, Jerry and Jack are equal partners. They decide to sell the house for $200:

- The partnership's inside basis in the house is $120.
- The FMV is $200.
- Tax liability: $80 (FMV of $200 minus inside basis of $120)

This $80 gain is allocated between Jerry and Jack equally. Even though Jerry paid $200 for his interest and economically feels like there is no gain, he still has a tax liability of $40.

**§754 Elections and Basis Adjustments** To correct this mismatch, when a partnership interest is sold, a §754 election allows the partnership to adjust the inside basis of its assets to match the buyer's contribution. The §754 election will trigger a basis adjustment under §743b to ensure that the inside and outside bases align, accurately reflecting the new partner's investment.

**Example of Sale of Partnership Interest with a § 754 Election** Building on our earlier example, suppose that when Jill sold her partnership interest to Jerry for $200, they made a §754 election. This would step up the inside basis of the house to $200. Now:

- The partnership's inside basis in the house is $200 (adjusted due to the §754 election).
- The FMV is $200.
- Tax liability: $0 (FMV of $200 minus inside basis of $200)

In this case, Jerry does not have to pay any tax on the sale of the house since his inside basis was stepped up to match his outside basis through the §754 election.

### 2.1.1 iBOB Scheme

The Installment Bogus Optional Basis (iBOB) scheme takes advantage of the §754 election to artificially step up the inside basis of partnership assets to eliminate tax liability. The iBOB scheme is considered tax avoidance because it does not meet the economic substance doctrine, which requires that transactions have a legitimate business purpose beyond just reducing taxes (U.S. Congress 1954).

The basic steps of the iBOB scheme are illustrated in Figure 2.2. Initially, Mr. Jones has a 99% partnership in both Jones Co. and Family Trust. Jones Co., in turn, has a 99% partnership in New Co. New Co. owns a hotel with a FMV of $200 and an inside basis of $120 that they wish to sell to Mr. Brown (Figure 2.2a).

If New Co. were to sell the hotel directly to Mr. Brown, Mr. Jones would be taxed on the difference between the hotel's FMV and his share of the inside basis. So, his taxable gain would be around $79 ($200 FMV*0.99 share-$120 inside basis, Figure 2.2a).

To avoid paying this tax, Family Trust can purchase Jones Co.'s partnership interest in New Co. using a promissory note with a face value of $199 (Figure 2.2b) and a §754 election. Importantly, because the promissory note is essentially a loan within Mr. Jones's network of controlled entities, it will never actually be repaid. With the §754 election, this transfer triggers a basis adjustment under §743(b), stepping up the inside basis of the hotel to $199, nearly equal to the hotel's fair market value.

Following the basis step-up, New Co. sells the hotel to Mr. Brown for $200 (Figure 2.2c). Since the inside basis has been increased to nearly match the sales price, the taxable gain is essentially eliminated for Mr. Jones.

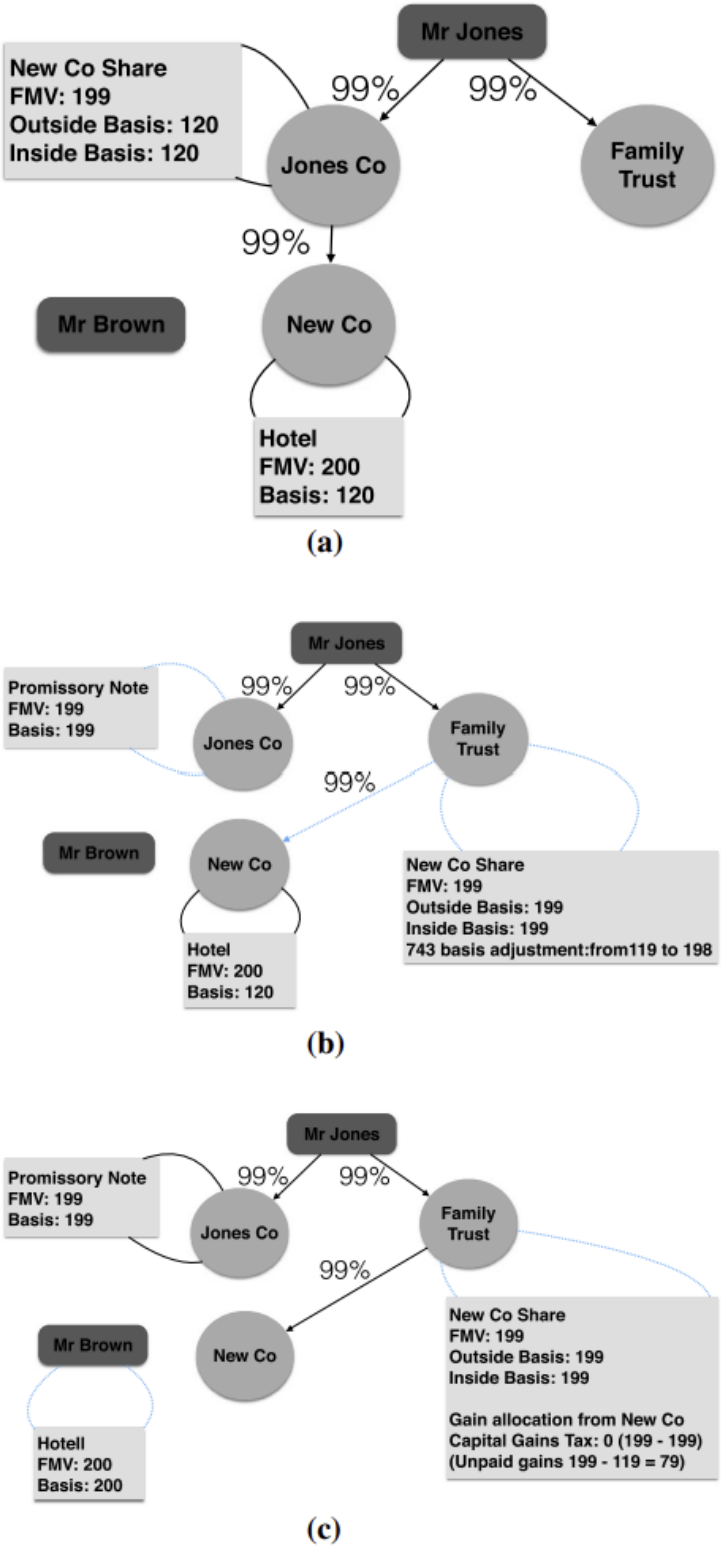


Figure 2.2: Tax Network with iBOB Scheme (Hemberg et al. 2016)

**Notes:** Installment Bogus Optional Basis (iBOB) scheme: (a) Mr. Jones's tiered ownership of a hotel with $200 FMV and $120 basis; (b) Family Trust buys New Co.'s interest for a $199 promissory note and makes a §754 election, stepping the hotel's inside basis to $199; (c) New Co. sells the hotel for $200, leaving virtually no taxable gain.

## 2.2 Preliminary Study

We begin by evaluating GPT-4o's ability to propose transactions and discover avoidance schemes through a series of prompt-based experiments. We focused on five core tasks: (1) finding tax avoidance strategies (e.g., iBOB); (2) finding iBOB within a specified tax network; (3) transaction formatting and ranking; (4) assessing legality; and (5) calculating tax. The findings, summarized in Table 2.1 below, shaped the design of the LLM-Tax Planner and LLM-Auditor.

| **Question** | **Results** | **Takeaways** |
|---|---|---|
| Can GPT-4o generate tax avoidance strategies? (e.g. see Sec. 2.2) | Yes. Given the tax code, GPT-4o proposed 20 strategies, one of which strongly resembled iBOB. | GPT can identify complex schemes, likely through memorization or pattern recognition. Since prompts were neutral, this suggests the ability is inherent and that mentioning tax avoidance in prompts could introduce bias. |
| Can GPT-4o discover iBOB in a given tax network? (e.g. see Sec. 2.2) | Only with guidance. GPT failed to reproduce iBOB in a fixed network without examples but succeeded after a one-shot prompt. | GPT needs scaffolding (e.g., example-based prompts or step-by-step guidance) to identify tax avoidance schemes in constrained settings. |
| Can GPT-4o format and rank transactions? (e.g. see Sec. 2.2) | Yes. GPT formatted transactions and ranked iBOB highest for both tax benefit and low suspiciousness. | GPT can formalize and evaluate schemes effectively but may prioritize the most subtle and aggressive strategies. |
| Can GPT-4o assess legality? (e.g. see Sec. 2.2) | No. GPT gave contradictory answers and often misclassified iBOB as legal, even when shown the exact sequence. | GPT lacks consistent legal reasoning and should not be used to determine legality; it is better suited for pattern detection. |
| Can GPT-4o calculate taxes correctly? (e.g. see Sec. 2.2) | No. GPT miscalculated basis and tax liabilities and made incorrect assumptions. | A rule-based calculator is needed for accurate tax computation. |

Table 2.1: Summary of Preliminary Results with GPT-4o

**Finding Tax Avoidance Strategies** Given the full text of Subchapter K, GPT-4o was prompted to identify potential tax minimizing strategies within partnership networks. It returned 20 distinct strategies, each with detailed transaction steps. One of these had a strong resemblance to the iBOB scheme, though it is unclear whether this was a result of

independent reasoning or prior exposure to iBOB during training. When prompted further on that example, GPT reproduced the iBOB transaction sequence (e.g. see Figure 2.3). This suggests two key takeaways: (1) the model is capable of surfacing complex and potentially aggressive tax strategies with alarming clarity, either by generalization or recall; and (2) to avoid biasing results, the prompts given to the tax planner had no mention of tax avoidance or iBOB.

3. **Contributed Property and Basis Adjustment Rules**

- Example Steps:
  - **Step 1**: A partner contributes property with a low tax basis (high unrealized gain) to the partnership.
  - **Step 2**: The partnership holds the property, benefiting from the appreciated value without realizing gains.
  - **Step 3**: The partnership then elects a Section 754 basis adjustment upon a partner's death or exit, allowing the new partner's basis to reflect current market value and reducing taxable gain upon sale.
  - **Alternative**: The partnership distributes the appreciated property to another partner who can defer or offset gains, avoiding an immediate tax hit.
  - **Goal**: This approach defers taxes and allows partnerships to benefit from "stepped-up" basis adjustments, thereby reducing taxable gain when assets are eventually sold.

Figure 2.3: GPT-4o's iBOB Discovery

**Notes:** GPT-4o initially proposed 20 tax minimizing schemes. When further prompted, GPT-4o elaborated this strategy 3 into a fully realized iBOB-style transaction sequence, suggesting either latent knowledge or strong pattern recognition.

**Finding iBOB within a Specified Tax Network** After confirming that GPT-4o could recognize iBOB within the tax code, the next step was to test whether it could suggest iBOB within a specific tax network. To evaluate this, GPT-4o was prompted with the Mr. Jones network example (e.g. see Figure 2.2), and asked to propose transactions within that network to minimize tax liability. With this zero-shot prompt, GPT-4o did not reproduce the iBOB steps on its own, unless significantly guided by follow-up prompts.

However, when given a one-shot prompt, using the Mr. Jones example as a training case, and asked to perform the same task on a structurally similar network, it immediately generated a transaction sequence resembling iBOB. This result suggests that GPT-4o struggles to generate tax-minimizing, constrained transaction sequences within a fixed network unless guided by prior examples or a human expert. These findings motivated two design choices for prompting

the LLM-Tax Planner: first, breaking down the problem by prompting one transaction at a time rather than full transaction sequences, and second, incorporating a fitness function, since GPT-4o could suggest transactions but consistently struggled to conceptualize which ones actually minimized tax.

**Transaction Formatting and Ranking** Next, after GPT-4o identified transactions, it was able to (1) structure them into a specific format, Transaction(entity1, entity2, asset1, asset2), and (2) rank them based on both tax liability and how suspicious they looked. The model ranked the iBOB transaction as the best (e.g. see Figure 2.4). This is concerning because GPT-4o easiliy suggested the least suspicious tax avoidance scheme for the user, which also happens to be illegal. These results showed that our LLM-Tax Planner could be prompted to generate transactions in a formal structure and could potentially rank transactions relative to one another.

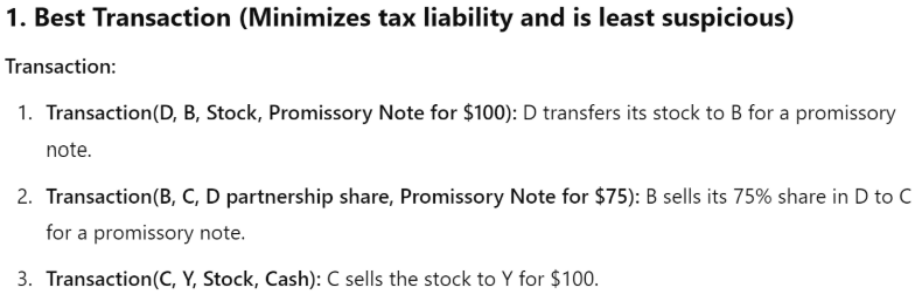


Figure 2.4: GPT-4o iBOB Discovery and Ranking

**Notes:** GPT-4o ranked transactions it proposed by both suspiciousness and tax liability, and picked the iBOB transaction sequences as the best. This raises concerns about its ability to prioritize "low-risk" avoidance strategies for users.

**Assessing Legality** After identifying the transaction steps within the iBOB tax scheme, the next question was whether GPT-4o would recognize these steps as illegal. The responses were contradictory. Initially, it stated that the iBOB transactions appeared legal but carried some risk. When asked to elaborate, it reversed its position, classifying them as lacking economic substance. Once this point was established, GPT-4o changed its stance on legality, now labeling the scheme as potentially abusive (e.g. see Figure 2.5).

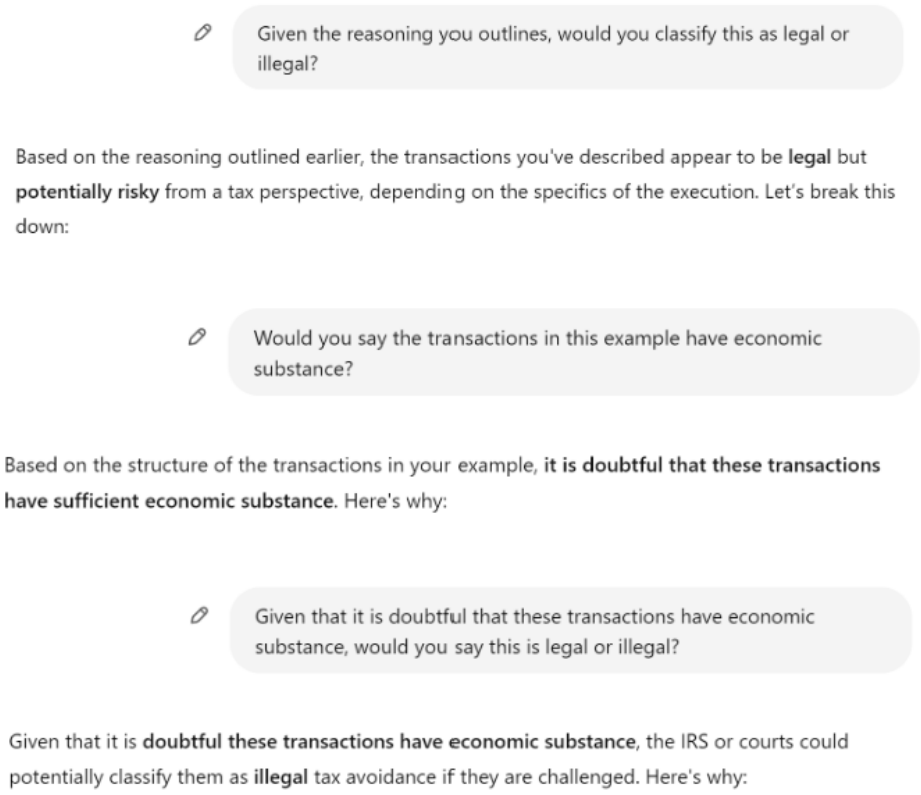


Figure 2.5: GPT-4o Contradicting Legal Assessment of Scheme

**Notes:** GPT-4o initially stated the scheme was legal but risky, then revised its judgment after prompting about economic substance.

However, when asked about the Mr. Jones iBOB steps explicitly, GPT-4o confidently claimed that the transactions were legal, did not resemble iBOB, and had economic substance, all of which are false (e.g. see Figure 2.6). These results are concerning. Even when GPT-4o organically generates a tax avoidance scheme within a network, it fails to recognize the scheme as illegal and often provides overly confident, yet incorrect legal assessments. This insight was important in designing the LLM-Auditor to not make legal determinations for each transaction, but to instead identify patterns, a task for which GPT seemed better suited.

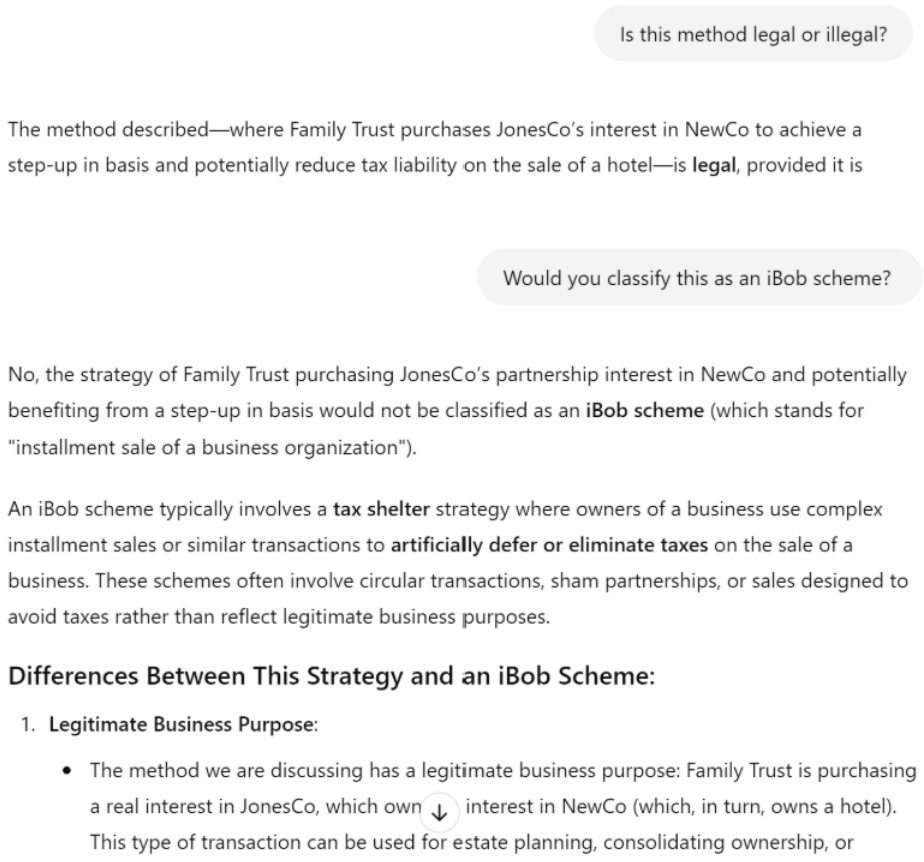


Figure 2.6: GPT-4o Misclassification of iBOB as Legal

**Notes:** When shown the iBOB sequence directly, GPT-4o incorrectly concluded the steps were legal, non-abusive, and economically substantial. Note GPT-4o incorrectly thinks that the iBOB acronym stands for installment sale of a business organization.

**Calculating Tax** Lastly, when GPT-4o was asked to calculate tax liability for different partners after transactions occurred, it was confidently wrong. It miscalculated inside and outside basis, made incorrect assumptions about ownership shares, and generally performed the calculations incorrectly. This reinforced the need for a separate tax calculator.

## 2.3 Related Work

To contextualize these results and motivate our approach, we briefly review prior work on tax planning, legal NLP, and evolutionary optimization.

**Tax Planning & Evolutionary Search** Previous studies of tax avoidance rely on computational models to show the dynamic between taxpayers and auditors. For example, an agent-based simulation by Bloomquist (2011) shows that individuals adjust their behavior after an audit. Extending this, Hemberg et al. (2016) use genetic algorithms to track how avoidance tactics coevolve with shifting enforcement rules in the iBOB scheme. Similarly, Rosen (2015) analyzes the Distressed Asset Debt scheme by mapping the movement of income and deductions through partnerships. Finally, Scott et al. (2024) use evolutionary search to optimize tax-planning strategies and note that this approach becomes harder to scale in more complex tax environments. Taken together, these studies demonstrate how computational tools can help us understand the mechanics of tax avoidance.

**Tax, Law & NLP** Natural language processing opens additional avenues for tax and legal analysis. Early work such as Pertierra et al. (2017) attempted to translate Internal Revenue Code provisions into default logic, illustrating both NLP's promise for tax research and the way statutory complexity still overwhelms conventional parsers. Building on a 376-case SARA dataset of simplified tax statutes, researchers show that GPT-3 frequently misreads even toy statutes, and that GPT-4 answers only about one-third of the cases exactly right (Holzenberger, Blair-Stanek, and Durme 2020; Blair-Stanek, Holzenberger, and Durme 2023 ; Blair-Stanek, Holzenberger, and Durme 2024). Dahl et al. (2024) add that large models often invent rulings or misattribute opinions, especially in lower-court cases. To mitigate these shortcomings, researchers are experimenting with role-based prompting; when S. Jung and J. Jung (2025) cast the model as specific courtroom actors (judge, clerk, advocate), they recorded measurable gains on statutory reasoning and other legal tasks. Even so, NLP techniques are most reliable when limited to supporting roles such as judgment prediction and document summarization (Zhong et al. (2020)).

**Tax Avoidance & NLP** Building on this line of work, Blair-Stanek, Holzenberger, and Durme (2022) sketch "Shelter Check," a system in which an LLM reads the entire tax corpus and tests every draft statute, ruling, or opinion to see whether it accidentally opens a new tax shelter. Extending that idea, Fratrič, Holzenberger, and Amariles (2025) translate statutes into a simple logical language and feed them into an agent-based simulator that explores thousands of corporate structures. Their prototype automatically rediscovered the "Double Irish with the Dutch Sandwich" scheme, showing that an LLM-plus-simulation combo can reveal real-world tax loopholes.

**Evolutionary NLP Pipelines** Improving the reasoning of large models remains an open challenge. Lee et al. (2025) proposed Mind Evolution, an iterative search method that refines model outputs and outperforms standard sampling on planning tasks. Similarly, Liventsev et al. (2023) introduce the SEIDR framework, which uses a feedback loop to test, identify, and correct errors in generated code. Though developed for programming, SEIDR's logic-driven refinement process holds promise for legal NLP. These evolutionary pipelines could enhance legal and tax applications by systematically improving reasoning through iterative feedback. Table 2.2 below summarizes this related literature.

**Addressing the Research Gap** We extend earlier agent-based and evolutionary work on tax planning (Hemberg et al. 2016; Scott et al. 2024) by replacing hand-coded agents with two large-language-model agents: one that proposes transaction sequences and one that audits them. Inspired by findings that LLMs often misread statutes (Pertierra et al. 2017; Blair-Stanek, Holzenberger, and Durme 2023; Dahl et al. 2024), we use a manual tax calculator. Unlike one-shot detection frameworks (Fratrič, Holzenberger, and Amariles 2025), our planner and auditor evolve in alternating rounds, allowing enforcement rules to adjust as avoidance tactics change. The coevolutionary schedule is similar in spirit to iterative refinement (Lee et al. 2025; Liventsev et al. 2023) but applied to two agents with opposing goals. Preliminary tests on the iBOB network show that the planner can uncover, and the auditor can flag, known avoidance paths, suggesting that this framework offers a practical way to study adaptive behavior on both sides of the tax-compliance problem.

| Authors | Paper | Evolution | NLP | Tax/Law | Avoid. |
|---|---|---|---|---|---|
| **Tax Planning & Evolutionary Search** | | | | | |
| Bloomquist (2011) | Tax Compliance as an Evolutionary Coordination Game: An Agent-Based Approach | Yes | No | Yes | Yes |
| Hemberg et al. (2016) | Detecting Tax Evasion: a Coevolutionary Approach | Yes | No | Yes | Yes |
| Scott et al. (2024) | Realistic Tax Planning with Evolutionary Algorithms | Yes | No | Yes | Yes |
| Rosen (2015) | Computer-Aided Policy Analysis of the DAD Scheme | Yes | No | Yes | Yes |
| **Tax, Law & NLP** | | | | | |
| Pertierra et al. (2017) | Towards Formalizing Statute Law as Default Logic through Automatic Semantic Parsing | No | Yes | Yes | No |
| Holzenberger, Blair-Stanek, and Durme (2020) | A Dataset for Statutory Reasoning in Tax Law Entailment and Question Answering | No | Yes | Yes | No |
| Blair-Stanek, Holzenberger, and Durme (2023) | Can GPT-3 Perform Statutory Reasoning? | No | Yes | Yes | No |
| Blair-Stanek, Holzenberger, and Durme (2024) | OpenAI Cribbed Our Tax Example, But Can GPT-4 Really Do Tax? | No | Yes | Yes | No |
| Dahl et al. (2024) | Hallucinating Law: Legal Mistakes with Large Language Models Are Pervasive | No | Yes | Yes | No |
| Zhong et al. (2020) | How Does NLP Benefit Legal System: A Summary of Legal Artificial Intelligence | No | Yes | Yes | No |
| S. Jung and J. Jung (2025) | Courtroom-LLM: A Legal-Inspired Multi-LLM Framework for Resolving Ambiguous Text Classifications | No | Yes | Yes | No |
| **Tax Avoidance & NLP** | | | | | |
| Blair-Stanek, Holzenberger, and Durme (2022) | Shelter Check: Proactively Finding Tax Minimization Strategies via AI | No | Yes | Yes | Yes |
| Fratrič, Holzenberger, and Amariles (2025) | Can AI expose tax loopholes? Towards a new generation of legal policy assistants | No | Yes | Yes | Yes |
| Bhattacharya | Discovering and Detecting Tax Avoidance Using Natural Language Processing and Coevolutionary Algorithms | Yes | Yes | Yes | Yes |
| **Evolutionary NLP Pipelines** | | | | | |
| Lee et al. (2025) | Evolving Deeper LLM Thinking | Yes | Yes | No | No |
| Liventsev et al. (2023) | Fully Autonomous Programming with Large Language Models | Yes | Yes | No | No |

Table 2.2: Overview of Related Work by Method and Focus

**Notes:** The first two columns list the authors and paper title. Evolution marks papers that use an evolutionary or other iterative-search method. NLP indicates explicit use of natural-language-processing or large-language-models. Tax/Law flags a primary focus on tax law or broader legal analysis, and Avoid notes an explicity emphasis on tax-avoidance or evasion schemes.

# 3

# Methodology

We begin by defining a structured tax environment that includes a set of entities, assets, and partnerships (e.g. see Sec. 3.1). Within this environment, we introduce two large language model (LLM) agents: a tax planner that proposes sequences of transactions to reduce tax liability, and an auditor that identifies patterns that could signal suspicious transactions. These two agents evolve in response to each other over multiple generations.

Figure 1.1 provides an overview of our coevolutionary framework. The environment sits at the center, maintaining the state of the tax network, actions, and outcomes. The tax planner generates transaction sequences based on the current tax network, while the auditor generates patterns based on the resulting transactions. Both agents evolve their outputs across generations, adapting in response to each other. The actions of these agents are then fed back into the environment, updating the network and being evaluated by the tax calculator.

Each agent works with its own type of representation. The tax planner generates full transaction sequences, where each transaction in sequence involves asset transfers between entities (e.g. see Sec. 3.2). The auditor, on the other hand, creates generalizable patterns that capture features of potentially suspicious transactions (e.g. see Sec. 3.2). We use a rule-based tax calculator to evaluate the tax consequences of each sequence, and the auditor labels each sequence with the patterns it identified.

We then define a fitness function for each agent to guide its evolution (e.g. see Sec. 3.3). The tax planner's fitness depends on both tax savings and the ability to avoid patterns flagged by the auditor. The auditor's fitness depends on how often its patterns appear across the transaction sequences and how often these patterns tag suspicious activity.

To evolve, each agent goes through a selection and mutation process. The tax planner modifies parts of its sequences to improve tax outcomes while avoiding audit suspicion (e.g.

see Sec. 3.3). The auditor builds new patterns from older ones to better capture frequent and suspicious features among the transactions (e.g. see Sec. 3.3). The agents evolve in an alternating cycle: first the tax planner adapts to avoid the previous generation's audit patterns, and then the auditor updates its patterns based on the newly generated transactions (e.g. see Sec. 3.3). This setup lets us observe how strategies and detection methods coevolve over time. We elaborate on these methods below.

## 3.1 Environment

To define the environment for our LLM-Agents, we initialize a dynamic tax network modeled as a directed graph $G = (V, E, A, T)$, where:

- $V$ is the set of entities (e.g., taxpayers, companies),
- $E \subseteq V \times V \times [0, 1]$ is the set of directed, weighted edges representing partnerships, with each edge $(v_i, v_j, w)$ indicating that entity $v_i$ holds a partnership interest of share $w$ in $v_j$,
- $A$ is the set of assets,
- $T$ is the set of transactions.

Each transaction $\tau \in T$ is defined as:

$$\tau = \{v_s, a_s, v_r, a_r, \S754\},$$

denoting a transfer of asset $a_s$ from entity $v_s$ to entity $v_r$, possibly accompanied by a §754 election.

To evaluate the tax liability of these transactions, we use a tax calculator that tracks ownership changes and calculates resulting tax. Together, the network, the set of actions, and the calculator create a full pipeline: beginning with an initial tax network, we can generate transactions, update the network based on those actions, and measure tax outcomes. This environment provides the structure both agents rely on: the tax planner to propose new sequences, and the auditor to identify patterns within these transactions. Figure 3.1 below provides an overview of this environmental pipeline.

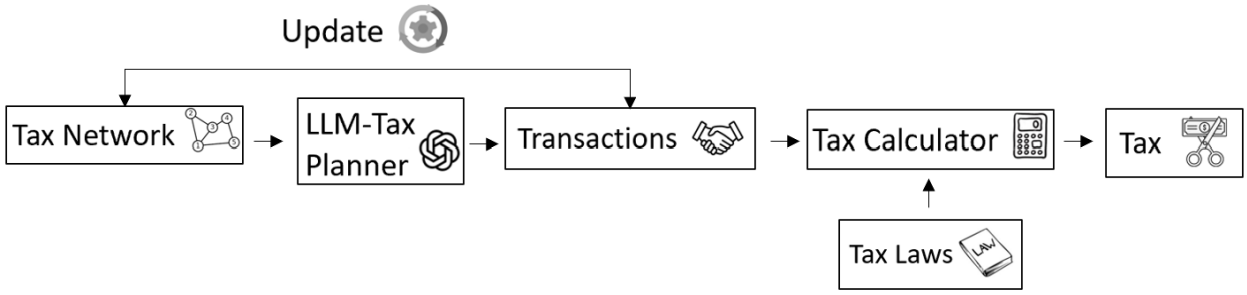


Figure 3.1: Overview of Environmental Pipeline

**Notes:** The current tax network feeds the tax planner, which generates a sequence of transactions; these transactions update the network and pass through a rule-based tax calculator that applies tax laws to compute the resulting liability, completing the feedback loop for the next round.

**Initial Network** The initial tax network is defined as a directed graph $G = (V, E, A, T)$, shown in Figure 3.2, with:

$$V = \{\text{Mr. Jones}, \text{Jones Co.}, \text{New Co.}, \text{Family Trust}, \text{Mr. Brown}\}$$
$$A = \{\text{Hotel}, \text{Cash}, \text{Promissory Note}, \text{Partnership}\}$$

The hotel (basis \$120, FMV \$200) is initially owned by New Co. Each entity holds \$200 in cash and a promissory note worth \$200. Partnerships, represented as directed edges, are:

$$E = \{(\text{Mr. Jones}, \text{Jones Co.}, 0.99), (\text{Mr. Jones}, \text{Family Trust}, 0.99), (\text{Jones Co.}, \text{New Co.}, 0.99)\}.$$

These relationships may be restructured through transactions in $T$.

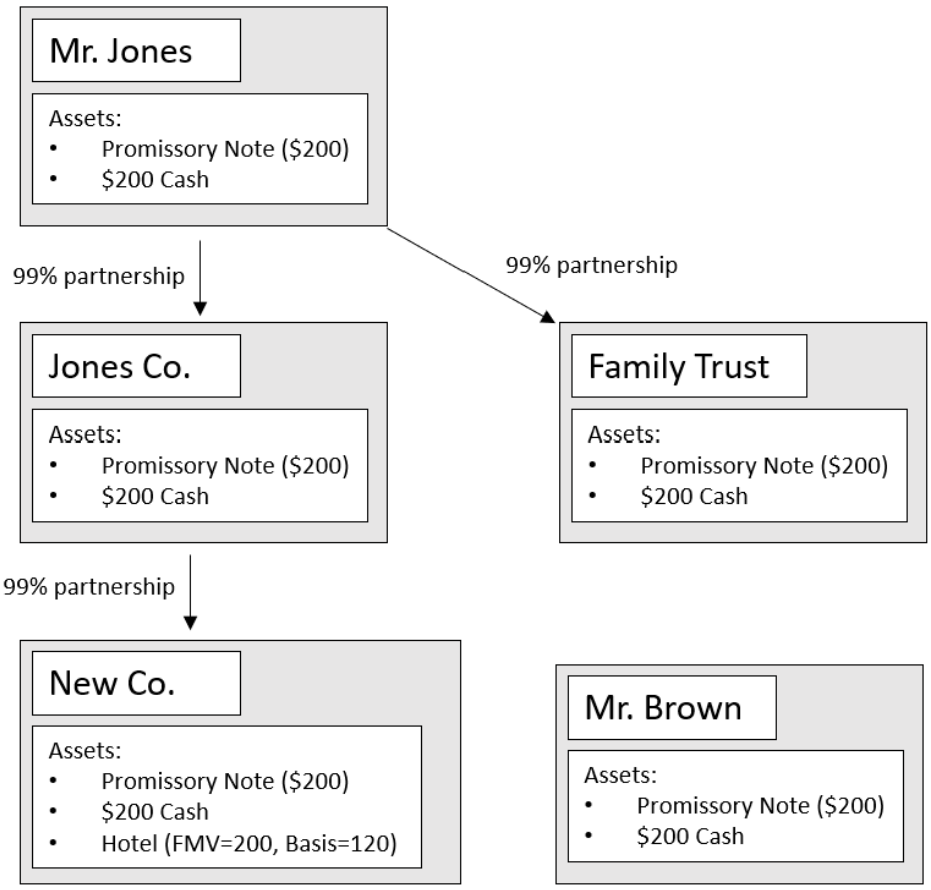


Figure 3.2: Graph G: Initial Tax Network

**Notes:** Mr. Jones holds 99% interests in both Jones Co. and the Family Trust, while Jones Co. owns 99% of New Co. Every entity starts with a $200 promissory note and $200 cash, and New Co. additionally owns the hotel (FMV $200, basis $120).

**Tax Actions** Once the tax network is fully specified, we represent transactions using a transaction sequence. A transaction sequence is an ordered list of transactions that occur one after the other. Each transaction involves one entity transferring an asset to another entity in exchange for a different asset. A §754 election may optionally accompany the transaction, set as a Boolean. Formally, a transaction is represented as:

```
Transaction(Entity1, Asset1, Entity2, Asset2, §754)
```

For example, an iBOB transaction sequence might look like:

```
[Transaction(Jones Co, Partnership w/New Co., Family Trust, $200, §754),
  Transaction(New Co., hotel, Mr.  Brown, $200, no §754)]
```

In this sequence, the first transaction has Jones Co. transferring its partnership interest with New Co. to the Family Trust in exchange for $200, with a §754 election in place. In the second transaction, New Co. sells the hotel to Mr. Brown for $200. It is the job of our tax planner to generate these transaction sequences.

**Tax Calculator** To assess the tax implications of such transaction sequences, we developed a tax calculator based on our understanding of the underlying tax law (e.g. see Sec. 2.1). This calculator processes the entire sequence within the defined network, updates the ownership status within the network, and computes the total tax liability for all entities involved.

## 3.2 Transaction and Pattern Generation

To generate transactions, we first use the LLM-Tax Planner to create valid sequences of transactions that move ownership of the hotel to Mr. Brown, while respecting all network constraints. Then, to create patterns, the LLM-Auditor reviews the transactions to identify common suspicious behavior and labels each transaction sequence with these patterns. This gives us a labeled set of transaction sequences that we can systematically analyze for both tax and audit risk.

**Transaction Generation** With this method to represent transactions, we now prompt our tax planner to generate a population of transaction sequences. Each sequence must be valid and must result in Mr. Brown getting ownership of the hotel.

To produce an individual sequence, we first select a random integer between two predefined hyperparameters: the minimum and maximum number of transactions in a sequence. This randomly chosen number determines the transaction sequence length. We then prompt the tax planner to generate the first transaction based on the original ownership status of the network as defined in Figure 3.2.

This prompt, shown in Appendix B.1.1, defines the conditions for a valid transaction. It prohibits, for example, violating the ownership status of the network by transferring assets an entity does not own or exceeding available cash balances. Importantly, the prompt includes no reference to the iBOB scheme and does not suggest our ultimate goal of transferring ownership of the hotel to Mr. Brown. Instead, the prompt simply asks the tax planner to propose creative transactions that are possible within the network.

After the first transaction is generated, the ownership status of all assets and entities is updated. This revised ownership state is then passed back into the next prompt, letting the tax planner condition its next transaction on the most current version of the network. These updates are needed to avoid ownership errors in subsequent transactions. However, if the tax planner produces an invalid transaction, the process restarts from the beginning of the sequence.

This procedure continues until the sequence reaches the target number of transactions. If, by that point, the hotel has not yet been transferred to Mr. Brown, we manually append a final transaction that transfers the hotel to Mr. Brown in exchange for cash.

Once the sequence is finalized, we pass it through our tax calculator to determine its total tax liability across the network. We repeat this entire procedure to generate a population of transaction sequences. Each one reflects a valid transaction sequence within the network, incorporates ownership updates, and ends with Mr. Brown owning the hotel. Figure 3.3 gives an overview of this procedure. The pseudocode for this method is also provided in Algorithm 1.

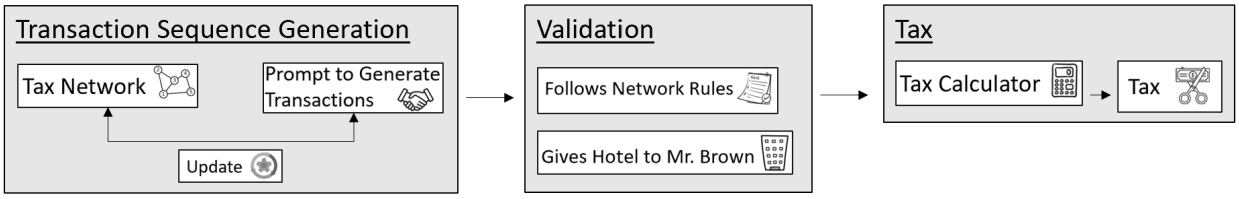


Figure 3.3: Generating Transaction Sequences

**Notes:** The tax planner proposes network-valid transactions; we manually verify that each sequence follows the rules and transfers the hotel to Mr. Brown, after which a tax calculator determines the resulting liability.

**Pattern Generation** Given the full set of transaction sequences produced by the tax planner with their associated tax amounts, we prompt the LLM-Auditor to generate an initial set of patterns that capture potentially suspicious transactions. We set a hyperparameter,

`num_patterns`, which determines how many distinct patterns the auditor should produce. Notably, we never define what a "suspicious" pattern is, leaving that to future work. The auditor has full freedom to define the structure and content of each pattern. The prompt used for generating patterns is provided in Appendix B.1.2.

With this set of patterns, we next prompt the auditor to label each transaction sequence with all applicable patterns. To avoid overwhelming the model, this process is done one sequence at a time. In each prompt, the auditor is given the full list of patterns along with a single transaction sequence, and asked to identify all relevant patterns in the sequence, with the requirement that the LLM identifies at least one pattern. This prompt can be found in Appendix B.1.2.

After applying this labeling procedure to the entire dataset, each transaction sequence generated by the tax planner is finally annotated with patterns identified by the auditor. Now, we are able to determine the frequency of each pattern across the entire transaction dataset. For reference, we define the frequency of a pattern as the number of transaction sequences in which the pattern appears divided by the total number of sequences (i.e., the population size). So, for example, a pattern with frequency 0.3 appears in 30% of the generated transaction sequences.

Next, we gauge the suspiciousness of each pattern by calculating the mean tax amount of the transactions it identifies. For instance, if a pattern mostly labels transactions with low tax, it will be considered more suspicious.

Now that we have the average tax amount and the frequency of a pattern, we can calculate that pattern's audit score. Better patterns will be more frequent and will have lower average tax.

With the labeled transaction data, we next find common co-occurrences, or unions, of patterns manually. For each transaction sequence, we compute all possible combinations of its identified patterns. For instance, if a sequence is identified having patterns 1, 2, and 3, we generate the combinations $1 \cup 2$, $2 \cup 3$, $1 \cup 3$, and $1 \cup 2 \cup 3$. From this, we obtain a full count of all co-occurring combinations of patterns within the entire transaction dataset.

With these combinations, we then filter out any duplicate co-occurrences. We then concatenate these co-occurring patterns with the original individual patterns to track all possible patterns. To keep the best solutions, we run a truncation selection on all these possible patterns and keep only `num_patterns` of those with the highest audit scores, individually and co-occurring.

The idea is to prioritize better patterns, either on their own or in combination with others. For consistency, we assign names and numeric identifiers to all final patterns. We then relabel each transaction sequence in the dataset with its updated pattern identifiers, including any matched co-occurring patterns.

With this final set of `num_patterns` identified, we now recalculate the frequency and average tax amount of our patterns to get a final audit score for every pattern.

At this stage, each of the patterns has an associated frequency, average tax amount, and audit score, and the transaction dataset is fully labeled with all detected patterns. Figure 3.4 gives an overview of this procedure. The pseudocode for this method is also provided in Algorithm 2.

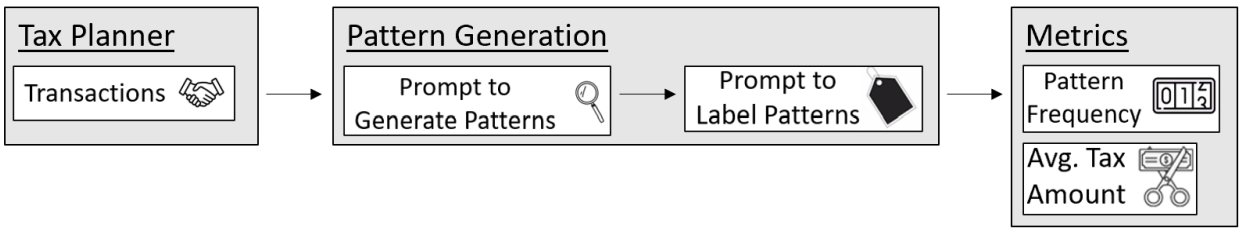


Figure 3.4: Generating Patterns

**Notes:** The tax planner's sequences feed the auditor, which first creates patterns, then tags each transaction with the matching patterns; pattern frequencies and average tax amount become pattern scoring metrics.

## 3.3 Transaction and Pattern Search

We describe how both the LLM-Tax Planner and LLM-Auditor evolve their solutions over time. Each agent selects high-performing candidates based on a fitness function, introduces mutations to create variations, and applies selection again to form the next generation. The two agents then coevolve in an alternating process, with the tax planner adapting to avoid detection and the auditor improving its ability to spot suspicious activity.

**Fitness**

Tax Planner Fitness: For a transaction sequence $t$, fitness $f_t$ combines tax liability and the

average audit score of the patterns that flag the sequence:

$$f_t \;=\; -\Big(C_t \,\mathrm{tax}(t) \;+\; \frac{1}{|P_t|}\sum_{p\in P_t} f_p\Big),$$

where $P_t$ is the set of patterns identified in transaction $t$ and $f_p$ is the audit fitness of pattern $p$. The coefficient $C_t$ rewards lower tax, while the second term penalizes sequences that look suspicious to the auditor.

Auditor Fitness: Each pattern $p$ is given fitness $f_p$ based on how often it appears and how low-tax its tagged transactions are:

$$f_p \;=\; \mathrm{freq}(p) \;-\; \frac{C_a}{|T_p|}\sum_{t\in T_p} \mathrm{tax}(t),$$

with $T_p$ the set of transactions labeled by pattern $p$. Higher frequency raises $f_p$; low tax pushes it down, making the pattern more "suspicious." The scaling factor $C_a$ makes patterns that label low-tax transactions have more of a suspicion penalty.

**Transaction Search** After generating an initial population of transaction sequences and assigning them fitness scores, the LLM-Tax Planner needs to evolve this population across generations toward higher fitness. To do this, we first perform a tournament selection to choose the transaction sequences with the highest fitness. This creates a new population of elite sequences the same size as the original population.

With this new population of elite transaction sequences, we then introduce mutations. We do this by randomly selecting a transaction sequence from the elite group and prompting the tax planner to mutate a randomly chosen transaction within that sequence. The only requirement is that the mutation must respect the ownership structure of the network. The tax planner is prompted to creatively choose whether to mutate an entity, an asset, or the §754 election status within the selected transaction. The prompt used for mutating transaction sequences is provided in B.1.1.

We repeat the mutation process `num_variations` times, a hyperparameter which determines how many transaction sequences to mutate. The resulting mutated sequences are then passed through our tax calculator to determine their total tax amounts. We combine these mutated transaction sequences with the original population of sequences and label all sequences with the audit patterns from the previous generation to calculate each sequence's average audit

scores, $\frac{1}{|P_t|}\sum_{p\in P_t} f_p$. Next, we calculate the fitness scores for both the original and mutated sequences. We then apply truncation selection to choose the top-performing transaction sequences to form the next generation. Figure 3.5 gives an overview of this evolution procedure. The pseudocode for this method is also provided in Algorithm 3.

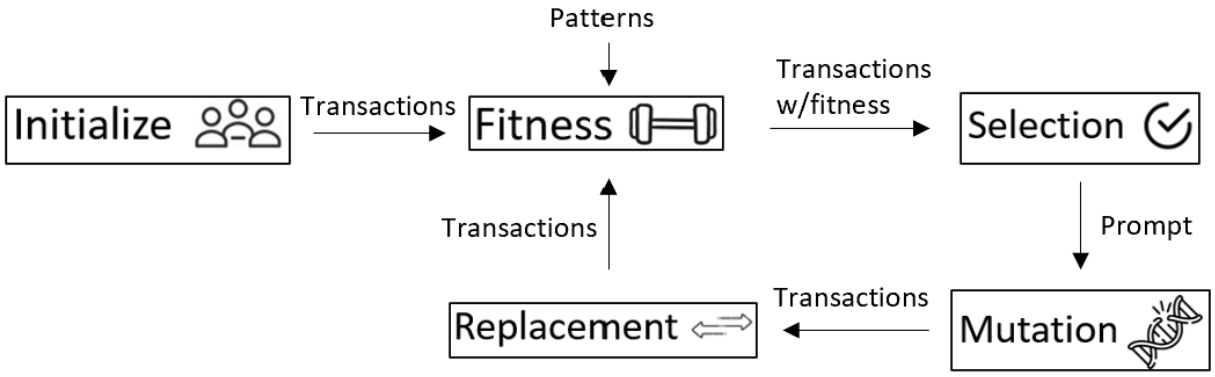


Figure 3.5: Transaction Evolution

**Notes:** An initial population of transactions is given fitness scores, tournament-selected elites undergo LLM-guided mutations, and the resulting variants replace the old population to form the next generation.

**Pattern Search** To evolve the LLM-Auditor's patterns, we start with a tournament selection, using fitness values from the previous generation of patterns. This produces a new population of high-performing patterns.

We then introduce variation, performing pattern mutation `num_variations` times. For each variation, we randomly select a pattern from this population and prompt the auditor for a mutated version. The goal is to build on existing patterns to create ones that generalize better, appear more frequently, and capture more suspicious activity in the dataset. The prompt for this mutation procedure can be found in B.1.2.

Once new variations are generated, we merge them with the original patterns, ensuring uniqueness. We relabel the transaction dataset using the full pattern pool to calculate each pattern's frequency and average tax amount. We also identify frequently co-occurring patterns, treating these unions as new composite patterns.

Fitness is then recalculated as a combination of pattern frequency and average tax amount. We then perform a truncation selection, retaining the top patterns for the next generation. Figure 3.6 gives an overview of this evolution procedure. The pseudocode for this method is also provided below in Algorithm 4.

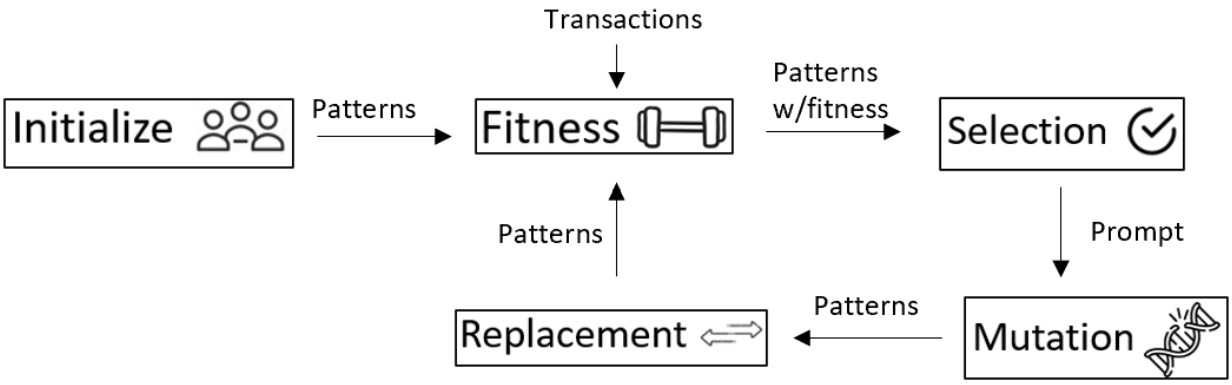


Figure 3.6: Pattern Evolution

**Notes:** Patterns are given fitness scores, tournament-selected, LLM-mutated into new variants, and the top performers replace the population for the next generation.

**Coevolution** Now that we have described how each LLM agent evolves on its own, we can explain how they coevolve by generating new populations of transaction sequences and audit patterns across generations.

After initializing both transaction sequences and audit patterns with their fitness scores (e.g. see Sec. 3.1), we evolve each agent using the outputs from the previous generation. At each generation, we begin by accessing the prior generation's transactions and patterns. The tax planner evolves first, optimizing for lower tax sequences that avoid detection by the previous generation's audit patterns (e.g. see Sec. 3.3). This results in a new set of transaction sequences.

Using this updated transaction population, the auditor then evolves to identify more frequent and suspicious patterns in the new data (e.g. see Sec. 3.3). It is important to note that the auditor evolves based on the current generation of transactions, while the tax planner evolves based only on the previous generation's patterns. This alternating coevolution gives the LLM-Auditor a slight advantage and is meant to reflect real-world dynamics, where auditors can review tax data before responding. Figure 1.1 gives an overview of the coevolution method. Algorithm 5 also gives the pseudocode for this method.

# 4

# Experiments

This section explores how large language models can both generate tax-planning transactions and audit them for suspicious activity. We explore three main components in our evaluation: the LLM-Tax Planner, which evolves transactions (e.g. see Sec. 4.1); the LLM-Auditor, which generates patterns to flag suspicious transactions (e.g. see Sec. 4.2); and a coevolving setup that combines both (e.g. see Sec. 4.3).

In the LLM-Tax Planner study we asked whether language models can generate and mutate sequences of transactions that both satisfy ownership rules and minimize tax. Structured prompts that enumerated the ownership rules cut semantic errors, but they also decreased solution diversity. In the evolutionary phase, the most iBOB schemes emerged when we mutated existing transactions rather than inserting or deleting them; however, this tactic increased semantic errors.

The LLM-Auditor experiments explored how prompt design affects an LLM's skill at identifying suspicious patterns. For the open-ended tasks of pattern generation and labelling, minimal prompts performed best, albeit with a modest rise in inconsistency. When the task switched to pattern mutation, where edits must respect earlier logic, more verbose prompts actually increased pattern diversity. Across runs, keeping the pattern set to four yielded the best trade-off between coverage and redundancy, and allowing patterns to co-occur led to better performance.

Finally, the coevolution experiments let the planner and auditor evolve together under different fitness functions. When the auditor's fitness rewarded only pattern frequency, the planner quickly converged on the iBOB scheme. Adding tax penalties to the auditor's score made the arms-race more realistic: the planner and auditor cycled between avoidance and detection instead of converging. Compared to benchmarks, coevolution yielding the highest fitness for each agent and promoted iBOB discovery. However, boosting the competitiveness of the auditor by increasing its number of patterns reduced iBOB domination.

## 4.1 LLM-Tax Planner

Table 4.1 presents the three experiments that structure the LLM-Tax Planner study: a transaction generation experiment (e.g. see Sec. 4.1.1), a transaction variation experiment (e.g. see Sec. 4.1.2), and a mutation-prompt sensitivity experiment (e.g. see Sec. C.1.1). Hyperparameters for each experiment are detailed in Table 4.2. We evaluate results based on the metrics described in Table 4.3. Collectively, our findings show that highly structured prompts sharply reduce semantic errors, while mutation-centric evolution is most effective at discovering low-tax transactions, though it does so with a modest loss in diversity and stability.

| Research Question | Experiments | Hypothesis | Results |
|---|---|---|---|
| Can an LLM generate transaction sequences that follow the rules of a tax network? | Transaction Generation (e.g. see Sec. 4.1.1) | A prompt that embeds ownership rules will produce better transactions than prompts relying solely on grammar or simplifications. | Partially affirms hypothesis: prompts with ownership rules have far fewer errors, at the cost of reduced diversity. |
| Can an LLM improve transaction sequences through evolution by using variations such as mutation, addition, and deletion? | Transaction Variation (e.g. see Sec. 4.1.2) | Using combinations of variations to evolve transactions will result in transactions with higher fitness and lower tax amounts. | Unexpected: the mutation variation is the most effective at discovering iBOB transactions, but leads to more errors during evolution. |
| How sensitive is an LLM-Tax Planner's performance to prompt design, particularly when varying prompt structure and content? | Transaction Generation (e.g. see Sec. 4.1.1), Mutation-Prompt Sensitivity (e.g. see Sec. C.1.1) | Using more verbose prompts, with clear, enumerated logic will lead to the best transaction generation and evolution. | Partially affirms hypothesis: more verbose generation and mutation prompts lead to fewer errors, at the expense of solution diversity. |

Table 4.1: LLM-Tax Planner Experiments

| **Hyperparameter** | **Exp 1: Transaction Generation** | **Exp 2: Transaction Variation** | **Exp 3: Mutation-Prompt Sensitivity** |
|---|---|---|---|
| Model | GPT-4o-mini | | |
| Number of Runs | 6 | | |
| Transaction Sequence Population Size | 100 | 10 | |
| Number of Generations | 1 | 10 | |
| Number of Transaction Variations per Generation | — | 8 | |
| Minimum Number of Transactions per Sequence | 2 | | |
| Maximum Number of Transactions per Sequence | 3 | | |
| Uniqueness Constraint | False | | |
| Number of Audit Patterns | — | 5 | |
| Number of Times a Sequence is Labelled with Patterns | — | 1 | |
| Type of Variation | — | All combinations tested | Mutation |
| Evolution of Auditor | — | No | |
| Evolution of Tax Planner | — | Yes | |
| Fitness Function | Fitness Function 1 (e.g. see 4.3.1) | | |

Table 4.2: Hyperparameters Used in LLM-Tax Planner Experiments

**Note:** Uniqueness constraint represents whether transaction sequences were required to be unique or not. The number of times a sequence is labeled with patterns indicates how many times the same sequence is labeled with the same set of patterns (the more times indicates desire for more labeling consistency). Evolution of auditor represents if the LLM-Auditor was evolved across generations, similar for evolution of tax planner. Fitness Function 1 was used in all experiments but became outdated in later coevolution experiments. The "—" indicates the hyperparameter is not applicable or was not specified for that experiment.

| Metric | Exp 1: Transaction Generation | Exp 2: Transaction Variation | Exp 3: Mutation-Prompt Sensitivity |
|---|---|---|---|
| Number of Errors | ✓ | | |
| Average Cosine Similarity of BERT Embeddings | ✓ | | |
| Number of Duplicates | ✓ | | |
| Average Fitness (Across Generations) | — | ✓ | |
| Average Fitness (Last Generation) | — | ✓ | |
| Number of iBOB Sequences Discovered | — | ✓ | |
| Tokens Used | ✓ | | |

Table 4.3: Evaluation Metrics Used in LLM-Tax Planner Experiments

**Note:** The "—" denotes that the metric is not used in an experiment.

### 4.1.1 T1: Transaction Generation

**Research Question and Experimental Design** Our first question asks: *Can an LLM generate transaction sequences that follow the rules of a tax network?* To answer this, we varied the transaction generation prompt and ran six independent trials per prompt, each producing 100 sequences. We evaluated results based on the metrics shown in Table 4.3, while the fixed hyperparameters appear in Table 4.2.

The prompts we tested were:

- **Verbose**: step-by-step numbered prompt with explicit ownership rules (Prompt B.1.1)
- **Minimal**: concise version of the verbose prompt (B.2.1)
- **Verbose Paragraph**: paragraph form of the verbose prompt (B.2.1)
- **Minimal Paragraph**: simplified paragraph version (B.2.1)
- **Grammar**: grammar-only prompt, no ownership status (B.2.1)
- **Verbose + Grammar**: Verbose prompt plus formal grammar (B.2.1)

- **Verbose + Grammar + Explanation**: previous prompt with explicit grammar explanation (B.2.1)

Several prompts embed a formal grammar that defines the correct syntax of a transaction sequence; for clarity, we reproduce that grammar in Figure 4.1.

```
<transactions> ::= <transactions><transaction> | <transaction>
<transaction> ::= Transaction(<entity>, <entity>, <asset>, <asset>, <election754>)
<entity>      ::= mr_jones | family_trust | new_co | mr_brown | jones_co
<asset>       ::= <cash> | <asset_name>
<cash>        ::= 200
<asset_name>  ::= promissory1_mrjones | partnership1 | partnership2 |
                  promissory1_jonesco | partnership3 |
                  promissory1_newco | hotel |
                  promissory1_familytrust | promissory1_mrbrown
<election754> ::= True | False
```

Figure 4.1: Formal Grammar Used in Select Generation Prompts

**Results and Analysis** As summarized in Table 4.4, the Verbose prompt yielded the lowest error rate while maintaining competitive diversity. Prompts that omitted ownership status produced the most errors, whereas hybrid prompts reduced errors only partially. Minimal or paragraph variants slightly improved diversity (fewer duplicates, lower tokens) but at the cost of correctness. Overall, the findings reinforce a core trade-off: tighter structure curbs errors but narrows the search space, while simpler prompts invite creativity but mistakes. Guided by our priority on reducing the number of errors, we adopt the Verbose prompt for subsequent experiments.

| Experiment | Number of Errors | Number of Duplicates | Cosine Sim. | Avg. Tokens per Sequence |
|---|---|---|---|---|
| Verbose | 18.7 (5.9) | 15.5 (4.8) | 0.9676 (0.0003) | 2687 (60) |
| Verbose + Grammar | 34.8 (6.4) | 17.7 (3.7) | 0.9689 (0.0012) | 3144 (113) |
| Verbose + Grammar + Explanation | 48.0 (6.9) | 9.2 (4.4) | 0.9656 (0.0008) | 3652 (120) |
| Minimal | 56.2 (18.0) | 9.0 (2.4) | 0.9662 (0.0014) | 1983 (97) |
| Verbose Paragraph | 57.8 (10.6) | 7.7 (3.9) | 0.9672 (0.0018) | 2200 (63) |
| Minimal Paragraph | 58.2 (14.6) | 8.5 (3.4) | 0.9685 (0.0011) | 1609 (29) |
| Grammar + Explanation | 275.2 (39.9) | 23.8 (4.0) | 0.9728 (0.0006) | 3343 (54) |
| Grammar | 345.7 (48.7) | 24.5 (4.0) | 0.9779 (0.0007) | 2632 (30) |

Table 4.4: LLM-Tax Planner, Transaction Generation Experiment

**Note:** Each row shows the mean across 6 runs for a prompt pattern. Columns show number of errors, duplicates, average cosine similarity for embeddings, and average tokens per sequence. Numbers are averaged across six runs with standard deviations shown in parenthesis. Best results highlighted for each column.

### 4.1.2 T2: Transaction Variation

**Research Question and Experimental Design** Our second question is: *Can an LLM improve transaction sequences through evolution by using variations such as mutation, addition, and deletion?* We fixed the evolutionary settings shown in Table 4.2 and ran six trials per variation strategy. We use the evaluation metrics noted in Table 4.3. Note, in this experiment, we used a prior, outdated fitness function. But since this fitness function was held constant across trials, relative comparisons are still valid.

The variation operators we explored were:

- **Add**: insert a new transaction.
- **Add and Delete**: delete one transaction, then insert a new one.
- **Mutate**: replace a transaction with a prompted mutation.

- **Add | Add + Delete**: randomly choose **Add** or **Add and Delete**.
- **Add | Mutate**: randomly choose **Add** or **Mutate**.
- **Add and Delete | Mutate**: randomly choose **Add and Delete** or **Mutate**.
- **Add | Add and Delete | Mutate**: randomly choose any of the three operators.

**Results and Analysis** Table 4.5 shows that the Mutate strategy uncovers the most iBOB-matching sequences and has the best average fitness, although it also makes the most mistakes. Combining Mutation with Add-and-Delete steps cuts the error rate somewhat, while relying only on Add or on Add + Delete finds fewer good sequences and lags in fitness. Every approach, however, improves its average fitness from one generation to the next, which means the selection step is working. Cosine-similarity scores and the number of duplicates barely change during the run. In short, focused mutation is still the quickest way to reach high-fitness, low-tax sequences, confirming that mutations are an effective way to search the iBOB space.

| **Exp.** | **Num iBOB** | **Num Errors** | **Avg. Fitness** | **Fitness Last Gen.** | **Cosine Sim.** | **Num Duplicates** | **Tokens** |
|---|---|---|---|---|---|---|---|
| Mutate | 25.83 (29.14) | 12.00 (7.69) | -101.43 (13.98) | -87.10 (39.87) | 0.95 (0.01) | 82.50 (2.17) | 238472 (9895) |
| Add and Delete, Mutate | 11.83 (24.34) | 12.83 (10.38) | -97.92 (25.49) | -84.11 (36.74) | 0.95 (0.01) | 74.67 (7.12) | 233217 (14728) |
| Add, Mutate | 7.17 (16.13) | 14.00 (7.80) | -111.53 (22.06) | -100.34 (41.89) | 0.95 (0.01) | 75.00 (10.28) | 233921 (6970) |
| Add, Add and Delete, Mutate | 5.17 (11.70) | 18.00 (6.20) | -104.72 (15.98) | -95.81 (28.79) | 0.95 (0.01) | 69.67 (8.07) | 234763 (10285) |
| Add | 2.17 (5.31) | 11.00 (4.15) | -119.97 (14.04) | -104.32 (30.22) | 0.95 (0.01) | 72.17 (5.31) | 456966 (18718) |
| Add, Add and Delete | 0.33 (0.82) | 8.67 (6.02) | -116.05 (17.48) | -115.47 (17.26) | 0.93 (0.01) | 77.33 (6.92) | 218143 (8850) |
| Add and Delete | 0.00 (0.00) | 6.67 (2.73) | -109.01 (13.37) | -103.07 (13.03) | 0.95 (0.02) | 81.50 (3.62) | 434268 (12203) |

Table 4.5: LLM-Tax Planner, Transaction Variation Experiment

**Note:** Each row shows the mean across 6 runs for a variation experiment. Columns include the number of iBOB transaction sequences discovered, total errors, average fitness across all generations, average fitness in the final generation, average embedding cosine similarity, number of duplicate transactions, and total tokens used. Numbers are averaged across six runs with standard deviations shown in parenthesis. Best result is highlighted in each column.

### 4.1.3 T3: Additional Experiments

**Mutation-Prompt Sensitivity Experiment** See Appendix C.1.1 for full details, but the key result is that detailed mutation prompt improves transaction evolution. The Verbose mutation prompt outperforms simpler versions on every accuracy measure, discovering more iBOB transactions, achieving higher fitness scores, and generating fewer errors. This replicates our transaction generation findings: rich, step-by-step instructions give up a little diversity in return for fewer errors.

## 4.2 LLM-Auditor

Table 4.6 presents the four experiments that structure the LLM-Auditor study: an initial pattern generation experiment (e.g. see Sec. 4.2.1), a mutation-prompt-sensitivity experiment (e.g. see Sec. 4.2.2), a number of patterns experiment (e.g. see Sec. C.2.1), and a co-occurring pattern experiment (e.g. see Sec. C.2.2). The hyperparameters in each experiment are detailed in Table 4.7, with evaluation metrics shown in Table 4.8.

We find that prompt wording has an effect on how the LLM-Auditor agent behaves. Concise, minimal prompts tend to maximize pattern frequency but introduce modest increases in labeling inconsistencies. In contrast, verbose prompts offer greater label stability and support richer mutation diversity, though at a slight cost to pattern frequency. This trade-off reflects a deeper principle grounded in the autoregressive objective of large language models:

$$P(y \mid x) = \prod_{t=1}^{T} P(y_t \mid y_{<t}, x),$$

where $x$ is the prompt (conditioning context), and $y_t$ is the token generated at time step $t$. Tasks like mutation, which rely on structured variation and conditional consistency, benefit from detailed prompts that effectively guide the model's output distribution. For more free-form tasks such as initial pattern generation or simple labeling, minimal prompts are often preferable because they reduce constraints and allow the model's prior $P(y)$ to have greater fluency. These findings tell us that optimal prompt style depends on a task's goals and how those goals interact with the model's internal biases.

Additionally, we find that varying the number of patterns reveals that four provides the

best balance between spread and coverage, and allowing co-occurring patterns yields a small yet consistent gain in both frequency and diversity. Together, these results highlight the importance of tailoring prompt design and pattern parameters to the specific demands of the LLM-Auditor's tasks.

| **Research Question** | **Experiments** | **Hypothesis** | **Results** |
|---|---|---|---|
| To what extent can an LLM detect and identify patterns in transaction data? | Pattern Generation (e.g. see Sec. 4.2.1) | Verbose, highly structured prompts generate the best, most general patterns (as measured by frequency). | Opposite: Minimal prompts generated the most frequently occurring patterns. Pattern frequency improved when we removed prompt guidance. However, minimal prompts increased labeling inconsistencies slightly. |
| Can an LLM improve audit patterns through evolution to better recognize suspicious transactions? | Pattern Variation (e.g. see Sec. 4.2.2), Number of Patterns (e.g. see Sec. C.2.1), Co-occurring Patterns (e.g. see Sec. C.2.2) | Verbose mutation prompts will yield higher pattern frequency. Using 5 patterns will best balance diversity and frequency. Allowing co-occurring patterns will improve both. | Partially affirms hypothesis: Verbose prompts create more diverse patterns but with slightly lower frequency. Four patterns struck the best balance. Co-occurring patterns improved pattern quality. |
| How sensitive is an LLM-Auditor's performance to prompt design, particularly when varying prompt structure and content? | Pattern Generation (e.g. see Sec. 4.2.1), Pattern Variation (e.g. see Sec. 4.2.2) | Verbose prompts will be best for pattern quality by guiding clearer structure. | Mixed: Minimal prompts performed better for pattern generation and labeling. Verbose prompts were more effective for pattern mutation by offering clearer guidance. |

Table 4.6: LLM-Auditor Experiments

| **Hyperparameter** | **Exp 1: Pattern Generation** | **Exp 2: Pattern Variation** | **Exp 3: Number of Patterns** | **Exp 4: Co-occurring Patterns** |
|---|---|---|---|---|
| Model | GPT-4o-mini | | | |
| Number of Runs | 6 | | | |
| Transaction Sequence Population Size | 100 | 10 | | |
| Number of Generations | 1 | 10 | | |
| Number of Pattern Variations per Generation | — | 3 | | |
| Minimum Number of Transactions per Sequence | 2 | | | |
| Maximum Number of Transactions per Sequence | 3 | | | |
| Number of Audit Patterns | 5 | | Varied (2–6) | 5 |
| Number of Times a Sequence is Labeled with Patterns | 3 | 1 | | |
| Pattern Unions Allowed | Yes | | | Varied |
| Uniqueness Constraint | False | | | |
| Evolution of Auditor | — | Yes | | |
| Evolution of Tax Planner | — | No | | |
| Fitness Function | Fitness Function 1 (e.g. see 4.3.1) | | | |

Table 4.7: Hyperparameters Used in LLM-Auditor Experiments

**Note:** "—" indicates the hyperparameter is not used in that experiment. In Exp 3, the number of audit patterns varied from 2 to 6. Pattern unions were allowed except in Exp 4 where they were explicitly manipulated. Evolution of auditor represents if the LLM-Auditor was evolved across generations, similar for evolution of tax planner. Fitness Function 1 was used in all experiments but became outdated in later coevolution experiments.

| Metric | Exp 1: Pattern Generation | Exp 2: Pattern Variation | Exp 3: Number of Patterns | Exp 4: Co-occuring Patterns |
|---|---|---|---|---|
| Average Audit Pattern Frequency | ✓ | | | |
| Average Audit Pattern Frequency (Last Generation) | — | ✓ | | |
| Average Number of Labeling Inconsistencies per Transaction Sequence | ✓ | — | | |
| Average Cosine Similarity of BERT Embeddings | ✓ | | | |
| Total Tokens Used | ✓ | | | |

Table 4.8: Evaluation Metrics Used in LLM-Auditor Experiments

**Note:** Checkmarks indicate that the metric was evaluated in that experiment. "—" indicates that the metric is not applicable to that experiment.

### 4.2.1 A1: Pattern Generation

**Research Question and Experimental Design** To investigate prompt sensitivity and pattern generation, our first question is: *To what extent can an LLM identify meaningful audit patterns, and how does prompt design influence that ability?* With the hyperparameters in Table 4.7 and the metrics in Table 4.8, we compared three prompt pipelines:

- **Verbose (Full)** – verbose prompt to generate patterns, verbose prompt to make these patterns clear (ClearPattern prompt), verbose prompt to label patterns (App. B.2.2–B.2.2);
- **Minimal (Full)** – concise versions of all three prompts (App. B.1.2, B.2.2, B.1.2);
- **Minimal (no ClearPattern)** – only the minimal generation and labeling prompts, omitting the clear pattern prompt.

**Results and Analysis** Table 4.9 indicates that the Minimal (no ClearPattern) prompt attains the highest pattern frequency (0.88) while using the fewest tokens, whereas the Verbose (Full) prompt yields the lowest rate of labeling inconsistencies (0.29 per sequence). Minimal prompts also exhibit the lowest cosine similarity, signifying greater pattern diversity. Because the slight rise in inconsistencies is acceptable once labeling repetitions are reduced from three to one, we adopt a two-prompt pipeline as the default. Illustrative patterns like *frequent transfers of assets between Mr. Brown and NewCo*, *promissory-note transactions between FamilyTrust and Mr. Brown*, and *repeated transactions executed without a §754 election* illustrate the interpretability of the auditor's patterns. Concise prompts maximize coverage and diversity, whereas verbose prompts exchange some of those gains for marginally higher consistency.

| **Experiment** | **Avg. Audit Frequency** | **Avg. Labeling Inconsistencies** | **Cosine Sim.** | **Total Tokens** |
|---|---|---|---|---|
| Verbose (Full) | 0.52 (0.11) | 0.29 (0.10) | 0.63 (0.01) | 142144 (3124) |
| Minimal (Full) | 0.79 (0.10) | 0.33 (0.15) | 0.61 (0.04) | 115700 (1734) |
| Minimal (no ClearPattern) | 0.88 (0.06) | 0.40 (0.07) | 0.62 (0.02) | 117605 (3183) |

Table 4.9: LLM-Auditor, Pattern Generation Experiment

**Note:** Each row corresponds to a prompt configuration. Reported metrics include: average pattern frequency (proportion of patterns matched per sequence), average labeling inconsistencies per sequence (out of 3 labels), cosine similarity between labeled outputs, and total tokens used. Numbers are averaged across six runs with standard deviations shown in parenthesis. Best result is highlighted in each column.

### 4.2.2 A2: Pattern Variation

**Research Question and Experimental Design** To further investigate prompt sensitivity and the evolution of our LLM-Auditor, we next ask: *How does mutation-prompt phrasing influence the evolution of audit patterns?* Using the evolutionary settings in Table 4.7 and the metrics in Table 4.8, we ran an experiment testing two mutation prompts:

- **Verbose** — verbose prompt (B.1.2);
- **Minimal** — concise prompt (B.2.2).

**Results and Analysis** Table 4.10 shows that the Minimal prompt has slightly higher audit-pattern frequency (0.97 overall and in the final generation) and lower token usage, while the Verbose prompt achieves the highest embedding dispersion (cosine similarity 0.74), indicating greater pattern diversity. Since diversity is more critical in the context of mutation, we adopt the Verbose mutation prompt for all subsequent work. Ultimately, this highlights a key principle: when creativity and variation are important, such as in mutation tasks, the verbose prompt is the better choice. Minimal prompts should be reserved for cases where diversity is secondary and constraints are tighter.

| **Experiment** | **Avg. Audit Frequency** | **Avg Audit Frequency Last Gen** | **Cosine Sim.** | **Total Tokens** |
|---|---|---|---|---|
| Verbose Mutation | 0.95 (0.05) | 0.96 (0.04) | 0.74 (0.04) | 113156 (4789) |
| Minimal Mutation | 0.97 (0.03) | 0.97 (0.05) | 0.79 (0.04) | 109953 (5166) |

Table 4.10: LLM-Auditor, Pattern Variation Experiment

**Note:** This table compares mutation prompt variants. Metrics include: average audit frequency, audit frequency on final generation, cosine similarity of outputs, and total tokens used. Numbers are averaged across six runs with standard deviations shown in parenthesis.

### 4.2.3 A3: Additional Experiments

**Number of Patterns Experiment** Table C.2.1 presents the full results for this experiment. Our key finding is that initializing the LLM-Auditor with four patterns provides the most extensive and least redundant coverage of the transaction space, as indicated by the lowest cosine similarity. We proceed with using four patterns in subsequent coevolution experiments.

**Co-occurring Patterns Experiment** Complete results are shown in Table C.2.2. Allowing patterns to co-occur produces a modest, consistent rise in audit frequency and slightly broader coverage, evidenced by a lower cosine similarity. Because of this, we allow co-occurring patterns in subsequent coevolution experiments.

## 4.3 Coevolution Experiments

To answer our coevolution research questions, we conduct three experiments: a fitness function comparison test (e.g. see Sec. 4.3.1), a comparison of coevolution results to benchmarks (e.g. see Sec. 4.3.2), and a richness of generations and patterns test (e.g. see Sec. 4.3.3). These experiments are summarized in Table 4.11, with experimental settings given in Table 4.12 and Table 4.13.

Overall, we find that when audit fitness rewards only pattern frequency, the LLM-Tax Planner quickly gravitates toward the tax-minimizing iBOB scheme. Adding tax penalties to the auditor's fitness slows this progression and introduces mild oscillations between the planner and auditor. Compared to baselines where only the auditor or the planner evolves, coevolution is crucial. With a fixed auditor, the planner quickly rediscovers iBOB, and with a fixed planner, the auditor merely tracks frequent patterns. Coevolution, on the other hand, instead pushes both agents towards higher fitness, iBOB discovery, and oscillatory behavior. We also see that increasing the number of patterns an auditor can detect improves their ability to distinguish iBOB from non-iBOB schemes, delaying iBOB dominance. This also leads to more frequent oscillations in planner behavior, reflecting a more dynamic and competitive coevolutionary process and suggesting the need for more specific audit patterns or iBOB penalties.

| **Research Question** | **Experiments** | **Hypothesis** | **Results** |
|---|---|---|---|
| How does the choice of a fitness function affect the convergence, oscillation, or diversity of transactions and patterns identified through coevolution? | Fitness Function Comparison (e.g. see Sec. 4.3.1) | Different fitness functions will influence convergence, oscillation, and iBOB strategy emergence; combining tax and frequency in audit fitness will yield more realistic dynamics. | Confirmed: Fitness function choice significantly affected outcomes. Frequency-only led to iBOB dominance; adding tax penalties led to more realistic oscillatory dynamics. |
| How does coevolution compare to benchmarks such as standalone pattern or transaction generation, sole LLM-Tax Planner evolution, or sole LLM-Auditor evolution? | Coevolution vs. Benchmarks (e.g. see Sec. 4.3.2) | Full coevolution will outperform benchmarks in discovering tax schemes and evolving effective patterns. | Confirmed: Only full coevolution discovered sophisticated iBOB variants and reached highest audit/planner fitness. Benchmarks showed partial abilities; Planner-only found iBOB but lacked adaptability; Auditor-only refined patterns but found no iBOB. |
| How does increasing the number of generations and audit patterns affect the long-run behavior of coevolution? | Generations and Pattern Richness (e.g. see Sec. 4.3.3) | More audit patterns reduce the number of iBOB transactions, delay iBOB dominance, and induce more oscillations by increasing detection capacity. | Confirmed: With 10 patterns, iBOB reduced, emergence delayed, and oscillations increased. Auditor pattern richness slowed convergence and promoted more adaptive dynamics. |

Table 4.11: Summary of Coevolution Experiments

| **Hyperparameter** | **Exp. 1: Fitness Function Comparison** | **Exp. 2: Coevolution vs. Benchmarks** | **Exp. 3a: Generations and Pattern Richness (4 patterns)** | **Exp. 3b: Generations and Pattern Richness (10 patterns)** |
|---|---|---|---|---|
| Model | GPT-4o-mini | | | |
| Number of Runs | 6 | | | |
| Number of Generations | 10 | Depends | 20 | |
| Transaction Sequence Population Size | 10 | | | |
| Minimum Number of Transactions per Sequence | 2 | | | |
| Maximum Number of Transactions per Sequence | 3 | | | |
| Number of Audit Patterns | 4 | | | 10 |
| Number of Audit Variations per Generation | 3 | | | 8 |
| Number of Tax Variations per Generation | 8 | | | |
| Pattern Unions Allowed | Yes | | No (computational limits) | |
| Number of Times a Sequence is Labeled with Patterns | 1 | | | |
| Transaction Generation Prompt | Verbose | | | |
| Transaction Mutation Prompt | Verbose | | | |
| Type of Transaction Variation | Mutation | | | |
| Pattern Generation Prompt | Minimal | | | |
| Pattern Labeling Prompt | Minimal | | | |
| Pattern Mutation Prompt | Verbose | | | |
| Uniqueness Constraint | True | | | |
| Evolution of Auditor | Yes | Varied | Yes | |
| Evolution of Tax Planner | Yes | Varied | Yes | |
| Fitness Function | Varied | Fitness Function 5 (e.g. see 4.3.1) | | |

Table 4.12: Hyperparameters Used in Coevolution Experiments

**Note:** We proceed with the optimal prompt types discovered in earlier experimentation. We ensure that transactions and patterns are unique within a generation for more precise results. We vary the evolution of each agent during experimentation.

| Metric | Exp 1: Fitness Function Comparison | Exp 2: Coevolution vs. Benchmarks | Exp 3: Generations and Pattern Richness |
|---|---|---|---|
| Best Fitness (Planner/Auditor) | - | ✓ | ✓ |
| Generation with Best Fitness (Planner/Auditor) | ✓ | ✓ | ✓ |
| Avg. Fitness (Planner / Auditor) | - | ✓ | ✓ |
| Average Audit Pattern Frequency | ✓ | ✓ | ✓ |
| Average Audit Tax Amount | - | ✓ | ✓ |
| Number of iBOB Transactions Discovered | ✓ | ✓ | ✓ |
| Number of iBOB Transactions Discovered per Generation | ✓ | ✓ | ✓ |
| Audit Fitness of iBOB Patterns | - | ✓ | ✓ |
| Audit Frequency of iBOB Patterns | ✓ | ✓ | ✓ |
| Audit Tax Amount iBOB Patterns | - | ✓ | ✓ |
| Oscillation Count | ✓ | ✓ | ✓ |

Table 4.13: Evaluation Metrics Used in Coevolution Experiment

**Note:** A checkmark indicates the metric was used to evaluate the experiment; '—' means it was not. Average audit pattern frequency measures how often a pattern appears across transactions. Audit tax amount is the mean tax value of transactions flagged by a pattern. iBOB-specific metrics apply only to patterns that label iBOB transactions. Oscillation count refers to full cycles where iBOB appears, disappears, and reappears over consecutive generations.

### 4.3.1 C1: Fitness Function Comparison

**Research Question and Experimental Design** We begin by experimenting with our fitness functions to address the research question: *How does the choice of a fitness function affect the convergence, oscillation, or diversity of transactions and patterns identified through coevolution?* We find that the choice of fitness function plays an important role in shaping the kinds of solutions discovered, influencing the emergence and detection of iBOB as well as

determining whether solutions tend toward oscillation or convergence.

**Audit Fitness** Audit fitness refers to the fitness of a given pattern generated by the LLM-Auditor. Broadly speaking, we designed audit fitness to reflect the level of suspicion associated with a pattern: the higher the audit fitness, the more suspicious the pattern. Our goal is to identify patterns that maximize suspiciousness. To this end, we explore two key metrics: frequency and the average tax amount of the transactions that a pattern labels. The rationale is that more frequent patterns are more likely to be suspicious. Therefore, audit fitness increases with pattern frequency, measured as the proportion of times a pattern appears in the planner's transaction sequences. In addition, we consider the average tax amount of the transactions flagged by a given pattern. As this average tax amount decreases, the level of suspicion increases, meaning that audit fitness is inversely related to tax amount.

**Planner Fitness** Planner fitness measures the fitness of a transaction sequence generated by the LLM-Planner. We focus on two main parameters to evaluate it: tax and audit fitness. First, with respect to tax, the planner tries to discover tax-minimizing schemes, so the fitness of a transaction sequence increases as the total tax amount decreases. Second, we incorporate audit fitness by calculating the average audit fitness of all patterns identified within a given transaction sequence. Since audit fitness represents suspicion, planner fitness is designed to decrease as audit fitness increases. In other words, as the average suspicion associated with a transaction sequence rises, its overall fitness should fall.

**Fitness Functions Tested** With these key concepts in mind, we begin by experimenting with fitness functions. For each function, we coevolved our planner and auditor for 10 generations, with each generation having 4 audit patterns and 10 transaction sequences, see metrics and hyperparameters in Table 4.13 and Table 4.12. We observe that the choice of fitness function significantly shapes the behavior of each agent. Below, we provide a detailed description of each of the five fitness functions we tested. Additionally, Table C.4 and Table C.5 show how fitness is calculated for example transactions and patterns under each of these fitness functions.

Fitness 1: Baseline Audit Frequency Only

**Planner Fitness:**

$$f_t = -\left(\text{tax}(t) + \frac{1}{|P_t|}\sum_{p \in P_t} f_p\right)$$

where $P_t$ is the set of patterns labeling transaction $t$, and $f_p = \text{freq}(p)$ is the audit fitness of pattern $p$, and tax(t) is the tax amount of transaction $t$.

**Audit Fitness:**

$$f_p = \text{freq}(p)$$

*Motivation:* This baseline version isolates the influence of pattern frequency on planner behavior. Audit fitness is determined purely by how often a pattern appears, and tax is unscaled. This allows later comparisons with fitness functions that include scaling or tax-sensitive penalties.

Fitness 2: Linearly Scaled Audit Frequency

**Planner Fitness:**

$$f_t = -\left(\text{tax}(t) + \frac{1}{|P_t|}\sum_{p \in P_t} f_p\right)$$

where $P_t$ is the set of patterns labeling transaction $t$, and $f_p = 100 \cdot \text{freq}(p)$ is the scaled audit fitness of pattern $p$, and tax(t) is the tax amount of transaction $t$.

**Audit Fitness:**

$$f_p = 100 \cdot \text{freq}(p)$$

*Motivation:* This version tests how scaling up the influence of audit frequency affects planner behavior.

Fitness 3: Tax Incorporated into Audit Fitness

**Planner Fitness:**

$$f_t = -\left(\frac{1}{|P_t|}\sum_{p\in P_t} f_p\right)$$

where $P_t$ is the set of patterns labeling transaction $t$, and $f_p$ is the audit fitness of pattern $p$ as defined below.

**Audit Fitness:**

$$f_p = \text{freq}(p) - \frac{1}{|T_p|}\sum_{t\in T_p} \text{tax}(t)$$

where $T_p$ is the set of transactions labeled by pattern $p$ and tax(t) is the tax amount of transaction $t$.

*Motivation:* This function incorporates tax directly into audit fitness, flagging patterns that frequently label low-tax transactions. It tests whether embedding tax sensitivity into audit behavior changes the coevolution dynamics.

Fitness 4: Linearly Scaled Tax in Audit Fitness

**Planner Fitness:**

$$f_t = -\left(\frac{1}{|P_t|}\sum_{p\in P_t} f_p\right)$$

where $P_t$ is the set of patterns labeling transaction $t$, and $f_p$ is the audit fitness of pattern $p$ as defined below.

**Audit Fitness:**

$$f_p = \text{freq}(p) - \frac{1}{|T_p|}\sum_{t\in T_p} 100\cdot\text{tax}(t)$$

where $T_p$ is the set of transactions labeled by pattern $p$ and and tax(t) is the tax amount of transaction $t$.

*Motivation:* This function amplifies the suspicion for labeling low-tax transactions by scaling the tax component. It is designed to test how a stronger audit sensitivity to tax avoidance affects planner adaptation.

Fitness 5: Scaled Tax in Both Audit and Planner Fitness

**Planner Fitness:**

$$f_t = -\left(2 \cdot \text{tax}(t) + \frac{1}{|P_t|} \sum_{p \in P_t} f_p\right)$$

where $P_t$ is the set of patterns labeling transaction $t$, and $f_p$ is the audit fitness of pattern $p$ as defined below, and tax(t) is the tax amount of transaction $t$.

**Audit Fitness:**

$$f_p = \text{freq}(p) - \frac{1}{|T_p|} \sum_{t \in T_p} 100 \cdot \text{tax}(t)$$

where $T_p$ is the set of transactions labeled by pattern $p$ and tax(t) is the tax amount of transaction $t$.

*Motivation:* This function incorporates tax into planner fitness to more resemble real-life incentives. It explores how direct tax weighting in both audit and planner fitness affects coevolution.

**Results and Analysis**

Planner and Auditor Solutions: With every fitness function, the LLM-Planner discovered an iBOB-style transaction around generations three or four. Interestingly, rather than merely recreating the canonical iBOB example in Figure 2.2, the LLM additionally discovered a even better scheme:

(a) Mr. Jones transfers `partnership2` to `JonesCo` for `promissory1_jonesco`, electing §754.

(b) He exchanges `partnership1` with Mr. Brown for `promissory1_mrbrown`, again electing §754.

(c) `NewCo` swaps `promissory1_newco` for `partnership2` with `JonesCo`, invoking a third §754 election.

(d) `NewCo` sells the `hotel` to Mr. Brown for $200 *without* a §754 election.

The three elections here raise the hotel's inside basis to $200 while Mr. Jones's outside basis remains approximately $198, yielding an immediate $2 loss for Mr. Jones every time there is a sale of partnership with a §754 election.

This shows that our mutation strategy was able to exploit our tax calculator's weaknesses to discover a better scheme than previously thought possible. Besides demonstrating the

success of using LLMs to discover tax avoidance strategies within our framework, this results illustrates a potentially new use for LLMs in exposing weaknesses in tax calculators and expert knowledge.

Independently, the LLM-Auditor identified key patterns characteristic of iBOB schemes, demonstrating its ability to precisely detect and isolate indicators of such behavior. Below are some of the highest audit-fitness patterns among those that labeled iBOB transactions:

- Transactions involving repeated transfers of promissory notes or assets among limited partners, consistently structured with §754 elections, typically followed by recurring hotel transactions and occasionally resulting in negative tax amounts
- Repeated promissory note transactions with §754 elections leading to consistent subsequent hotel transactions without elections
- Use of promissory notes in transactions.

Fitness Function Comparison: Table 4.14 presents the outcomes of five coevolution trials. Overall, we find that when audit fitness is based solely on frequency, iBOB is still dominant, but the number of iBOB transactions minimally decreases as the penalty on frequency increases. Combining penalties for both low tax and high frequency in our audit fitness is more effective: planner and auditor fitness diverge and we occasionally see moderate oscillatory coevolution behavior. Reintroducing tax minimization in the LLM-Planner's fitness understandably increases iBOB re-emergence, yet the cyclical behavior somewhat remains.

Interestingly, we see that for fitness functions 3–5, LLM-Planner fitness declines over generations. This is likely because the auditor eventually labels most sequences as suspicious, making it hard to distinguish iBOB from non-iBOB schemes. As iBOB slightly outperforms non-iBOB, it becomes more prevalent across generations, raising overall audit fitness but lowering planner fitness. Since audit penalties are averaged, the signal weakens, explaining why we only sometimes see oscillatory shifts between iBOB and non-iBOB transactions.

To better understand these dynamics, we now examine each fitness function in detail. The breakdown below highlights how different audit and planner incentives shape coevolutionary behavior, including the emergence, suppression, and resurgence of iBOB strategies.

**Fitness 1:** When the planner's fitness relies almost entirely on tax minimization, the evolutionary search quickly discovers and converges on the iBOB strategy. As iBOB transactions proliferate, the auditor's frequency-based score rises monotonically because once every sequence looks like iBOB, frequency can only increase. The result is smooth, non-oscillatory growth in both agents' fitness and iBOB dominance (Figures C.3.2).

**Fitness 2:** Raising the penalty on common patterns makes frequent transactions more "suspicious," so iBOB no longer saturates the population as quickly as in Fitness 1, though it still eventually dominates. Planner and auditor fitness still climb steadily, and just as before, no meaningful cycling emerges, confirming that frequency alone is too weak a brake on iBOB prevalence (Figures C.3.2).

**Fitness 3** Adding an explicit penalty for low-tax outcomes reshapes the search landscape: iBOB sequences are now flagged as highly suspicious, trimming their overall count and yielding occasional oscillations (around 0.33 oscillations per run). Planner fitness falls while auditor fitness rises, characteristic of the inverse relationship typical of real-world enforcement dynamics (Figures C.3.2).

**Fitness 4** Increasing the tax-amount penalty by two orders of magnitude leaves the total share of iBOB transactions mostly unchanged yet increases oscillatory behavior. Stronger penalties encourage occasional shifts to non-iBOB transactions, but because audit patterns lack clear distinction, iBOB and non-iBOB still receive similar fitness, dampening the overall impact (Figures C.3.2)

**Fitness 5** Reintroducing a reward for tax savings for our tax planner increases iBOB incidence and dominance, although moderate oscillations remain. Planner fitness nevertheless declines across generations because the auditor's tax penalty component increases faster than tax rewards can compensate. This is why we see that planner fitness peaks at generation one, while audit fitness peaks at generation nine on average. However, this fitness function most closely models real world incentives and we still see a divergence of auditor and planner fitness (Figures C.3.2).

| Category | Metric | Fit. 1 | Fit. 2 | Fit. 3 | Fit. 4 | Fit. 5 |
|---|---|---|---|---|---|---|
| Planner Fitness | Generation with Best Fitness | 7.67 (2.34) | 6.0 (3.22) | 4.0 (3.46) | 3.0 (2.45) | 1.0 (0.0) |
| Audit Fitness | Average Pattern Frequency | 0.96 (0.03) | 0.98 (0.02) | 0.97 (0.03) | 0.97 (0.01) | 0.96 (0.05) |
| | Gen with Best Audit Fitness | 1.33 (0.52) | 1.17 (0.41) | 5.0 (4.52) | 7.0 (4.29) | 9.33 (1.21) |
| iBOB | Number of Total Transactions with iBOB Scheme | 54.17 (14.36) | 38.67 (24.29) | 12.83 (15.37) | 15.0 (23.95) | 38.17 (24.27) |
| | Avg Number of Transactions with iBOB Scheme per Generation | 6.98 (0.79) | 5.74 (3.02) | 2.56 (2.68) | 2.31 (2.56) | 5.35 (2.68) |
| | Generation Planner Found iBOB | 3.33 (1.37) | 4.4 (1.67) | 4.25 (2.5) | 3.5 (3.0) | 4.17 (2.32) |
| | Avg Frequency of iBOB Labeling Patterns | 0.99 (0.01) | 0.98 (0.03) | 0.92 (0.05) | 0.96 (0.04) | 0.95 (0.07) |
| Coevolution Dynamics | Number of Oscillations | 0.0 (0.0) | 0.0 (0.0) | 0.33 (0.82) | 0.67 (1.03) | 0.33 (0.82) |

Table 4.14: Coevolution, Coevolution Fitness Function Comparison Experiment

**Note**: **Red cells** mark row-wise **maximums**, **blue cells** mark **minimums**. Planner and audit fitness refer to their respective agents; iBOB metrics concern iBOB transactions. Coevolution dynamics capture oscillatory iBOB emergence behavior across generations. Numbers are averaged across six runs with standard deviations shown in parenthesis.

### 4.3.2 C2: Coevolution vs. Benchmarks

**Research Question and Experimental Design** We next addressed our second research question: *How does coevolution compare to benchmarks such as standalone pattern or transaction generation, sole LLM-Tax Planner evolution, or sole LLM-Auditor evolution?*. To gauge the contribution of coevolution under Fitness 5, we compared it with three benchmarks:

(1) Generation Only: ten transaction sequences and four audit patterns, with no evolution.

(2) Planner-only Evolution: the ten transaction sequences evolve for ten generations against four static patterns.

(3) Auditor-only Evolution: four patterns evolve for ten generations against ten static transaction sequences.

Across all runs we tracked average and best fitness, the generation of peak fitness, iBOB prevalence, and any oscillatory behavior (e.g. see Table 4.13 and Table 4.12).

**Results and Analysis** The results in Table 4.15 show that mutation and selection are essential for the planner to discover iBOB, which never appears without evolution. When only the planner evolves, iBOB quickly dominates but cannot stay fixed due to static audit pressure, leading to strong oscillations. When only the auditor evolves, tax avoidance remains flat while audit strategies steadily improve, showing the auditor's capacity for adaptation. But, coevolution pushes both agents to their highest fitness levels and peak iBOB discovery, outperforming any single-agent setup and producing the richest overall dynamics. We go into more detail below.

Generation Only: With no evolution, average planner fitness is highest since iBOB never appears. This absence highlights that mutation and selection are necessary for the LLM-Planner to uncover our avoidance scheme.

Planner-only Evolution: Because the auditor's four patterns never change, the planner quickly finds iBOB. The run shows the sharpest back-and-forth swings (Figures C.3.3). Early mutations quickly uncover iBOB, so the scheme dominates first. Over time, random drift and the fixed penalties make non-iBOB transactions competitive again, and the population swings back. Because nothing in the audit model adapts to fix either option in place, the system

keeps toggling between the high-savings/high-penalty iBOB peak and the safe-but-costly non-iBOB peak, producing the pronounced oscillations seen in Figures C.3.3.

Auditor-only Evolution: With transactions frozen, iBOB does not emerge; tax fitness stays flat, but audit fitness climbs each generation as patterns specialize (Figures C.3.3). The run confirms that the LLM-Auditor can evolve to generate patterns with higher fitness.

Full Coevolution (Fitness 5): Coevolution yields the highest peak tax fitness, the highest peak audit fitness, and the greatest number of iBOB sequences, along with moderate oscillations. Coevolution thus drives both agents to richer strategies than either baseline can reach in isolation.

| Category | Metric | Generation Only | Planner-only Evolution | Auditor-only Evolution | Coevolution |
|---|---|---|---|---|---|
| Planner Fitness | Avg. Fitness | 7839.04 (0.07) | 7462.51 (292.58) | 7839.03 (0.03) | 4640.68 (1970.45) |
| | Best Fitness | 7839.05 (0.08) | 7839.1 (0.09) | 7839.2 (0.23) | 7901.75 (84.84) |
| | Generation with Best Fitness | 1.0 (0.0) | 2.5 (2.81) | 2.0 (2.45) | 1.0 (0.0) |
| Audit Fitness | Avg. Audit Fitness | -7999.05 (0.08) | -7625.13 (292.46) | -7999.04 (0.04) | -4671.75 (1970.42) |
| | Avg. Tax Amount of Audit Patterns | 80.0 (0.0) | 76.26 (2.92) | 80.0 (0.0) | 46.73 (19.7) |
| | Avg. Freq of Audit Patterns | 0.95 (0.08) | 0.96 (0.03) | 0.96 (0.04) | 0.96 (0.05) |
| | Best Audit Fitness | -7999.0 (0.0) | -6539.01 (1451.42) | -7999.0 (0.0) | -272.65 (2053.7) |
| | Generation with Best Audit Fitness | 1.0 (0.0) | 4.83 (3.71) | 1.33 (0.52) | 9.33 (1.21) |
| iBOB | Number of Transactions with iBOB | 0.0 (0.0) | 5.67 (4.13) | 0.0 (0.0) | 38.17 (24.27) |
| | Avg Number of iBOB Transactions Per Gen | 0.0 (0.0) | 1.06 (0.7) | 0.0 (0.0) | 5.35 (2.68) |
| | Generation iBOB was First Discovered | – | 2.6 (0.89) | – | 4.17 (2.32) |
| | Avg Audit Fitness of Patterns Identifying iBOB Transactions | – | -6729.31 (756.52) | – | -2195.5 (2581.14) |
| | Avg Pattern Freq of Patterns Identifying iBOB Transactions | – | 0.96 (0.03) | – | 0.95 (0.07) |
| Coevolution Dynamics | Number of Oscillations | 0.0 (0.0) | 2.33 (3.14) | 0.0 (0.0) | 0.33 (0.82) |

Table 4.15: Coevolution, Coevolution vs. Benchmarks Experiment

**Note**: **Red cells** mark row-wise **maximums**, **blue cells** mark **minimums**. '–' entries are not applicable for comparisons. Planner and audit fitness refer to their respective agents; iBOB metrics concern iBOB transactions. Coevolution dynamics capture oscillatory iBOB emergence behavior across generations. Numbers are averaged across six runs with standard deviations shown in parenthesis.

### 4.3.3 C3: Generations and Pattern Richness

**Research Question and Experimental Design** We wanted to answer the research question: *How does increasing the number of generations and audit patterns affect the long-run behavior of coevolution?* In our earlier experiment on fitness functions, we found that despite some oscillations under fitness function 5, the population almost always converged on iBOB transactions. Our hypothesis was that planner fitness, being the average audit fitness (the mean suspicion assigned by every pattern labeling a transaction), gave iBOB and non-iBOB transactions almost identical scores whenever they were tagged with similar patterns. Because iBOB transactions had a lower tax liability, they gradually dominated the planner's solutions across generations.

We therefore reasoned that audit patterns able to discriminate more sharply between iBOB and non-iBOB transactions might disrupt this dominance. To test the idea, we ran two experiments, each for 20 generations. In the first, we held the audit set constant at four patterns per generation (as in the earlier study). In the second, we increased it to ten patterns per generation. All metrics and hyperparameters are summarized in Table 4.13 and Table 4.12. Finally, to visualize the dynamics over a longer horizon, we extended one run of each experiment to 60 generations.

**Results and Analysis** Results are shown in Table 4.16 below. The results confirm our hypothesis. When we only have four patterns, there are almost four times as many total iBOB transactions (88.83 compared to 24.5). We also see several more iBOB transactions per generation (6.06 compared to 1.94), suggesting that with more patterns, there is less iBOB domination. Raising the pattern count to ten cuts that dominance and more than doubles the number of oscillations, 2.83 versus 1.17. Understandably, as the number of audit patterns increases, average pattern frequency declines, planner fitness rises (due to fewer iBOB schemes and thus lower suspicion), and audit fitness falls for the same reason.

| Category | Metric | 4 Pattern Exp. | 10 Pattern Exp. |
|---|---|---|---|
| Planner Fitness | Avg. Fitness | 4004.91 (2350.74) | 6771.58 (1263.54) |
| | Best Fitness | 7839.14 (0.09) | 7839.27 (0.07) |
| | Generation with Best Fitness | 2.67 (2.73) | 4.67 (3.39) |
| Audit Fitness | Avg. Audit Fitness | -4011.87 (2400.24) | -6836.97 (1330.03) |
| | Avg. Tax Amount of Audit Patterns | 40.13 (24.0) | 68.38 (13.3) |
| | Avg. Freq of Audit Patterns | 0.95 (0.03) | 0.81 (0.1) |
| | Best Audit Fitness | 879.19 (408.96) | -3220.48 (4316.2) |
| | Generation with Best Audit Fitness | 18.83 (2.86) | 7.67 (5.75) |
| iBOB | Number of Transactions with iBOB | 88.83 (53.6) | 24.5 (30.02) |
| | Avg Number of iBOB Transactions Per Gen | 6.06 (2.68) | 1.94 (2.3) |
| | Generation iBOB was First Discovered | 5.67 (4.37) | 4.25 (2.06) |
| | Avg Audit Fitness of Patterns Identifying iBOB Transactions | -1392.47 (2347.97) | -4472.75 (2377.54) |
| | Avg Pattern Freq of Patterns Identifying iBOB Transactions | 0.95 (0.05) | 0.83 (0.08) |
| Coevolution Dynamics | Number of Oscillations | 1.17 (2.86) | 2.83 (3.13) |

Table 4.16: Coevolution, Generations and Pattern Richness Experiment

**Note**: Red cells mark row-wise maximums.

Visually, when we extend each experiment to 60 generations, a clear pattern emerges, as shown in Figure 4.5. In the run using just four audit patterns, we observe a brief period of oscillation betweeen iBOB and non-iBOB transactions, specifically, a single cycle between generations 0 and 10. However, by generation 20, iBOB becomes dominant and remains so, with average fitness for both agents stabilizing around that point. In contrast, when using ten patterns, we observe a total of four oscillations, with more frequent shifts and fluctuations. iBOB does not dominate until around generation 40, and we see continued variation in both agents' average fitness up to that point. Of course, we need to conduct more runs to rigorously assess these differences.

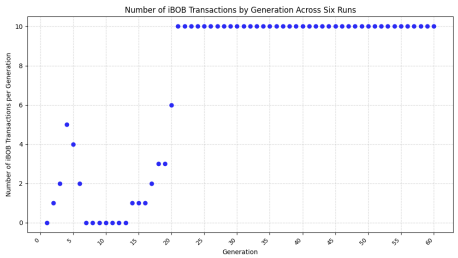


Figure 4.2: Four Patterns: Avg Number of iBOB Transactions per Generation Across Generations

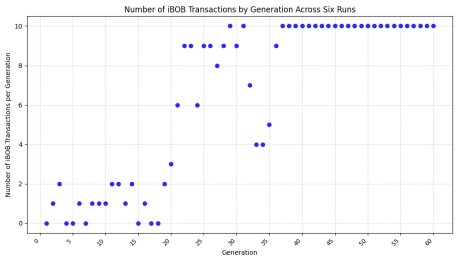


Figure 4.4: Ten Patterns: Avg Number of iBOB Transactions per Generation Across Generations

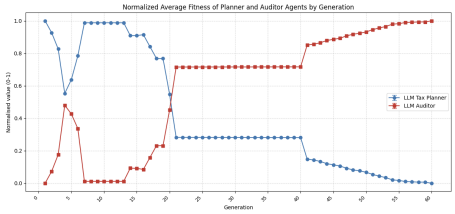


Figure 4.3: Four Patterns: Avg Fitness Across Generations

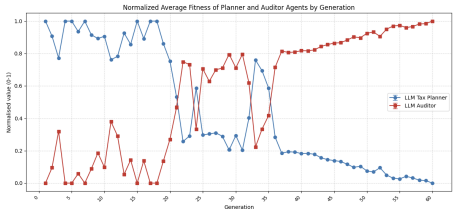


Figure 4.5: Ten Patterns: Avg Fitness Across Generations

These results support the idea that increasing the number of patterns the auditor can detect helps distinguish between iBOB and non-iBOB schemes, which in turn delays iBOB's dominance. This finding is especially interesting because in the real world, the number of patterns an auditor can recognize reflects a real constraint. Our experiment suggests that relaxing this constraint by giving auditors more capacity makes them more competitive in identifying and preventing tax avoidance. This aligns with reality: auditors with more resources tend to perform better. These results also raise a broader question about how other aspects of pattern design like quality might influence iBOB convergence. We leave that for future work.

# 5

# Conclusion

## 5.1 Summary

This thesis demonstrates that large language models (LLMs), when paired with well-designed prompts and coevolutionary algorithms, can generate, refine, and evolve complex transactions and audit patterns. Below, we summarize our findings for each component of our framework.

**LLM-Tax Planner**

- **Can an LLM generate transaction sequences that follow the rules of a tax network?**
  Yes. With structured, rule-based prompts, the LLM-Tax Planner generates transaction sequences that obey the rules of a tax network. Concise prompts increase transaction diversity but frequently violate network constraints. Explicitly encoding ownership rules in the prompt reduces semantic errors compared to concise prompts or those based solely on a grammar without embedded rules.

- **Can an LLM improve transaction sequences through evolution by using variations such as mutation, addition, and deletion?**
  Yes. Mutation alone is the most effective operator, giving the greatest number of iBOB transactions. Though, it slightly increases the number of semantic errors within mutated transactions.

- **How sensitive is an LLM-Tax Planner's performance to prompt design, particularly when varying prompt structure and content?**
  Performance is highly sensitive to prompt design. Detailed, itemized prompts increase accuracy and fitness, especially during mutation, whereas concise prompts create slightly more diverse transactions. When working within a well-defined tax network, structured

prompts are preferable because they better support precision and compliance with tax rules.

### LLM-Auditor

- **To what extent can an LLM detect and identify patterns in transaction data?**
  The model reliably detects recurring, interpretable audit patterns. Concise prompts produce high-frequency, broadly applicable patterns with slightly greater pattern labeling inconsistency, while verbose prompts improve consistency but reduce pattern frequency.

- **Can an LLM improve audit patterns through evolution to better recognize suspicious transactions?**
  Yes, given that the prompt style aligns with the task. In mutation tasks, verbose prompts generate a wider diversity of patterns. Also, initializing with four audit patterns balances coverage and redundancy, and allowing patterns to co-occur raises both diversity and pattern frequency. Our auditor-only evolution experiment also supports how the LLM-Auditor is able to improve audit fitness over generations.

- **How sensitive is an LLM-Auditor's performance to prompt design, particularly when varying prompt structure and content?**
  Very sensitive. Verbose prompts are more effective in mutation tasks, where structured variation is key, because they guide the model toward diverse, contextually rich patterns. In contrast, concise prompts work best for open-ended pattern generation, where flexibility and frequency matter more than structure.

### Coevolution

- **How does the choice of a fitness function affect the convergence, oscillation, or diversity of transactions and patterns identified through coevolution?**
  Fitness function design has a major impact. When audit fitness rewards only pattern frequency, the planner rapidly converges on iBOB strategies. Adding tax penalties to the auditor's fitness slows this convergence and introduces mild oscillations. The most realistic behavior emerges when tax considerations are embedded in both planner and auditor fitness: iBOB strategies are still discovered, but with ongoing fluctuations that

reflect real-world enforcement dynamics. Additionally, through coevolution, our auditor generates patterns very specific to iBOB characteristics while our tax planner discovers better iBOB variants than previously thought possible, highlighting an LLM's potential to reveal weaknesses in tax calculators or expert reasoning.

- **How does coevolution compare to benchmarks such as standalone pattern or transaction generation, sole LLM-Tax Planner evolution, or sole LLM-Auditor evolution?**
  Coevolution is essential. Without evolution, iBOB never appears. Planner-only evolution leads to iBOB discovery but triggers sharp oscillations due to static audit patterns. Auditor-only evolution does improve pattern fitness across generations, showing that even without iBOB transactions, auditor evolution can successfully find better patterns. But, only coevolution enables both agents to reach their highest fitness levels while also producing the oscillatory dynamics characteristic of adaptive behavior.

- **How does increasing the number of generations and audit patterns affect the long-run behavior of coevolution?**
  Expanding the auditor's capacity by increasing the number of audit patterns delays iBOB dominance and introduces more frequent oscillations. With ten patterns, coevolution is dynamic for longer. This shows us that the constraints on the auditor greatly contribute to evolutionary pressure. Regardless, extending the number of generations shows us the eventual convergence of iBOB within our current framework, suggesting that the auditor is still not competitive enough or our fitness function needs stronger iBOB penalties.

To summarize our key contributions, this thesis introduces a coevolutionary search framework in which two large language model agents adapt through successive cycles of mutation, evaluation, and selection. The LLM-Tax Planner evolves transaction sequences that minimize tax liabilities and mitigate audit risk, while the LLM-Auditor learns patterns that signal avoidance. Their continual interaction recreates the adversarial dynamics of tax enforcement. Applied to the iBOB scheme, the planner reliably uncovers existing iBOB strategies and even more effective variants, and the auditor learns to detect them, all within the constraints of a formal tax calculator. We find that the choice of prompt and fitness functions strongly shapes agent behavior and this evolutionary exchange.

By pairing a rule based tax calculator with iterative adversarial training, this work unifies traditional hand coded models of tax planning with recent single pass LLM audits. The result

is a platform for analyzing avoidance behavior, designing audit strategies, and informing tax policy.

## 5.2 Limitations

**Statistical Significance** In the future, running more trials could give enough data for robust statistical comparisons, making the results more rigorous.

**Computational Scale** Due to local compute limits, some trials used fewer generations or smaller populations than ideal. With larger compute budgets, future work could run longer experiments and sample the search space more thoroughly.

**Model Diversity** All of the experiments in this thesis relied on ChatGPT-4o-mini. But, other models can differ in how well they follow prompts, understand ownership rules, or learn effective audit patterns. Replicating the study with different LLMs would clarify how sensitive the framework is to model architecture and instruction tuning.

**Tax Calculator Accuracy** We used a hand-coded tax calculator to determine the tax liability of transactions. Although this guarantees that statutory rules are applied, it also risks hidden errors or unhandled edge cases, which could skew selection pressure or hide better avoidance strategies. Replacing or validating the calculator with a more rigorous or LLM-based-tax evaluator would improve accuracy.

**Other Avoidance Schemes** Our current tax calculator is specific to iBOB, which limited our ability to test other tax avoidance schemes. Future work could look into developing additional tax calculators, generalizing the existing one, or using an LLM-based calculator to see whether the results hold across different schemes.

## 5.3 Future Work

**Scaling Experiments** One next step is to scale the experiments by running more generations with bigger population sizes. We could also experiment more with hyperparameters like population size, the number or type of variations, and prompt verbosity. Longer trials could uncover behavior that shorter runs conceal.

**Suspicion Modeling** At present, the auditor flags transactions as suspicious based on their tax amount. A richer design would ask the LLM-Auditor to assign explicit suspicion scores and explain the reasons for each flag, introducing a more human like sense of doubt. This feedback will change how often certain patterns evolve and make them easier to interpret. Further, evolving the auditor so that it prefers patterns similar to known avoidance schemes, rather than ones that are merely common or low tax, should improve its ability to mutate patterns and detect evasion.

**Pattern Specificity and Audit Quality** Our experiments explored how pattern quantity affects coevolution dynamics, but future work should explore the role of pattern quality more in depth. One promising direction is to increase pattern specificity by grouping patterns into topical clusters. This could improve pattern quality by decreasing overlap.

**Real-World Constraints** Imposing capacity limits, such as capping the number or complexity of patterns the auditor generates or limiting how many avoidance schemes the planner proposes, will make the simulation more realistic and reveal how agents act when resources are scarce.

**LLM-Based Tax Reasoning** The current framework uses a fixed tax calculator to test compliance. A more ambitious goal is to train or prompt an LLM to infer tax liability directly from statutory text. This change would remove the need for hand written rules or human tax experts and allow the system to operate with avoidance strategies that are not yet encoded.

**Model Generalization and Robustness** Finally, repeating these experiments with several language model architectures will test the robustness of the observed coevolutionary dynamics and prompt engineering methods.

# References


Blair-Stanek, Andrew, Nils Holzenberger, and Benjamin Van Durme (Dec. 12, 2022). "Shelter Check: Proactively Finding Tax Minimization Strategies via AI". In: *Tax Notes Federal*. Available at SSRN: https://ssrn.com/abstract=4327110, p. 1515.

— (2023). *Can GPT-3 Perform Statutory Reasoning?* arXiv: 2302.06100 [cs.CL]. URL: https://arxiv.org/abs/2302.06100.

— (2024). *OpenAI Cribbed Our Tax Example, But Can GPT-4 Really Do Tax?* arXiv: 2309.09992 [cs.AI]. URL: https://arxiv.org/abs/2309.09992.

Bloomquist, Kim (2011). "Tax Compliance as an Evolutionary Coordination Game: An Agent-Based Approach". In: *Public Finance Review* 39.1, pp. 25–49. DOI: 10.1177/1091142110381640. eprint: https://doi.org/10.1177/1091142110381640. URL: https://doi.org/10.1177/1091142110381640.

Dahl, Matthew, Varun Magesh, Mirac Suzgun, and Daniel E. Ho (Jan. 2024). "Hallucinating Law: Legal Mistakes with Large Language Models Are Pervasive". In: *HAI, Stanford University*. URL: https://hai.stanford.edu/news/hallucinating-law-legal-mistakes-large-language-models-are-pervasive.

Espinosa, Javier Moreno and Alonso Carriles Alvarez (2022). "Artificial Intelligence and Its Application in the Study of the Legal Complexity of the Value Added Tax Act in Mexico". In: *Data Analytics Applications in Emerging Markets*. Ed. by José Antonio Núñez Mora and M. Beatriz Mota Aragón. Singapore: Springer Nature Singapore, pp. 177–202.

Fratrič, Peter, Nils Holzenberger, and David Restrepo Amariles (2025). *Can AI expose tax loopholes? Towards a new generation of legal policy assistants*. arXiv: 2503.17339 [cs.CY]. URL: https://arxiv.org/abs/2503.17339.

Guha, Neel et al. (2023). *LegalBench: A Collaboratively Built Benchmark for Measuring Legal Reasoning in Large Language Models*. arXiv: 2308.11462 [cs.CL]. URL: https://arxiv.org/abs/2308.11462.

Hemberg, Erik, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May O'Reilly (Apr. 2016). "Detecting Tax Evasion: a Co-Evolutionary Approach". In: *Artificial Intelligence and Law* 24.2. Version: Author's final manuscript, pp. 149–182. DOI: 10.1007/s10506-016-9181-y.

Holzenberger, Nils, Andrew Blair-Stanek, and Benjamin Van Durme (2020). *A Dataset for Statutory Reasoning in Tax Law Entailment and Question Answering*. arXiv: 2005.05257 [cs.CL]. URL: https://arxiv.org/abs/2005.05257.

Jung, Sangkeun and Jeesu Jung (Jan. 2025). "Courtroom-LLM: A Legal-Inspired Multi-LLM Framework for Resolving Ambiguous Text Classifications". In: *Proceedings of the 31st International Conference on Computational Linguistics*. Ed. by Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert. Abu Dhabi, UAE: Association for Computational Linguistics, pp. 7367–7385. URL: https://aclanthology.org/2025.coling-main.493/.

Katz, Daniel M. and Michael J. Bommarito (2014). "Measuring the complexity of the law: the United States Code". In: *Artificial Intelligence and Law* 22.4, pp. 337–374. DOI: 10.1007/s10506-014-9160-8. URL: https://doi.org/10.1007/s10506-014-9160-8.

Lee, Kuang-Huei, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen (2025). *Evolving Deeper LLM Thinking*. arXiv: 2501.09891 [cs.AI]. URL: https://arxiv.org/abs/2501.09891.

Legal Information Institute, Cornell Law School (2025). *26 U.S. Code Subchapter K — Partners and Partnerships*. Retrieved April 27, 2025. URL: https://www.law.cornell.edu/uscode/text/26/subtitle-A/chapter-1/subchapter-K.

Liventsev, Vadim, Anastasiia Grishina, Aki Härmä, and Leon Moonen (July 2023). "Fully Autonomous Programming with Large Language Models". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. ACM, pp. 1146–1155. DOI: 10.1145/3583131.3590481. URL: http://dx.doi.org/10.1145/3583131.3590481.

Pertierra, Marcos A., Sarah Lawsky, Erik Hemberg, and Una-May O'Reilly (2017). "Towards Formalizing Statute Law as Default Logic through Automatic Semantic Parsing". In: *ASAIL@ICAIL*. URL: https://api.semanticscholar.org/CorpusID:33245110.

Rappeport, Alan (Apr. 2021). "Tax cheats cost the U.S. $1 trillion per year, I.R.S. chief says$". In: *The New York Times*. URL: https://www.nytimes.com/2021/04/13/business/irs-tax-gap.html.

Rosen, Jacob (Jan. 2015). "Computer aided tax avoidance policy analysis". In.

Ruhl, J.B. and Daniel Katz (Nov. 2015). "Measuring, Monitoring, and Managing Legal Complexity". In: *Iowa law review* 101, pp. 191–244.

Scott, Eric O., Anmol Srivastava, Abby Pusateri, Andy Taylor, Karen Jones, and Hahnemann Ortiz (2024). "Realistic Tax Planning with Evolutionary Algorithms". In: *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. DOI: 10.1109/CEC60901.2024.10612201.

U.S. Congress (1954). *Definition of Economic Substance Doctrine, 26 U.S.C. § 7701(o)(5)(A)*. https://www.law.cornell.edu/definitions/uscode.php?width=840&height=800&iframe=true&def_id=26-USC-1846184229-1495237516&term_occur=2&term_src=title:26:subtitle:F:chapter:79:section:7701. Accessed: May 15, 2025.

Zhong, Haoxi, Chaojun Xiao, Cunchao Tu, Tianyang Zhang, Zhiyuan Liu, and Maosong Sun (July 2020). "How Does NLP Benefit Legal System: A Summary of Legal Artificial Intelligence". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault. Online: Association for Computational Linguistics, pp. 5218–5230. DOI: 10.18653/v1/2020.acl-main.466. URL: https://aclanthology.org/2020.acl-main.466.

# A

# Appendix A: Algorithms

## A.1 Generation Algorithms

**Algorithm 1** Generate Transaction Sequences

```
function GENERATE_TRANSACTIONS(population_size)
    results ← [ ]
    while length of results < population_size do
        ownership ← initial_ownership_state()
        target_length ← random_integer(MIN, MAX)
        sequence ← [ ]
        for i ← 1 to target_length do
            transaction ← LLM_GENERATE(ownership)
            if transaction is invalid then
                log error
                break and restart sequence
            else
                sequence.append(transaction)
                ownership ← UPDATE_OWNERSHIP(ownership, transaction)
            end if
        end for
        tax ← CALCULATE_TAX(sequence)
        results.append(sequence with tax)
    end while
    return first population_size results
end function
```

**Algorithm 2** Generate Audit Patterns, Labels, and Scores

```
function GET_AUDIT_SCORES(transactions, num_patterns)
    raw_patterns ← LLM_GENERATE_PATTERNS(transactions, num_patterns)
    clear_patterns ← LLM_CLARIFY_PATTERNS(raw_patterns)
    for each transaction in transactions do
        labels ← LLM_LABEL_TRANSACTION(clear_patterns, transaction)
        repeat labeling multiple times
        assign most frequent labels to transaction
    end for
    co_patterns ← FIND_CO_OCCURRENCE_PATTERNS(transactions)
    top_patterns ← TRUNCATE_TO_TOP(clear_patterns + co_patterns, num_patterns)
    pattern_frequencies ← COUNT_PATTERN_FREQUENCIES(transactions)
    for each pattern in top_patterns do
        suspicion_score ← avg_tax_amount
        audit_score = get fitness
    end for
    for each transaction do
        Audit Score ← sum of audit scores for its patterns
        Tax Fitness = get fitness
    end for
    return top_patterns, scored_transactions
end function
```

## A.2 Search Algorithms

**Algorithm 3** Evolve Transactions

```
function EVOLVE_TAX_TRANSACTIONS(initial_population, audit_patterns)
    elite_population ← TOURNAMENT_SELECTION(initial_population)
    mutated_sequences ← [ ]
    for i ← 1 to number_of_mutations do
        sequence ← RANDOMLY_SELECT(elite_population)
        mutated ← MUTATE_SEQUENCE(sequence)
        if MUTATION_RESPECTS_OWNERSHIP(mutated) then
            APPEND(mutated_sequences, mutated)
        end if
    end for
    combined_population ← elite_population + mutated_sequences
    LABEL_AUDIT_PATTERNS(combined_population, audit_patterns)
    ASSIGN_FITNESS_SCORES(combined_population)
    next_generation ← SELECT_TOP_PERFORMERS(combined_population)
    return next_generation
end function
```

**Algorithm 4** Evolve Audit Patterns

```
function EVOLVE_AUDIT_PATTERNS(patterns, transactions, num_variations)
    elite_patterns ← TOURNAMENT_SELECTION(patterns)
    new_patterns ← [ ]
    for i ← 1 to num_variations do
        pattern ← SELECT_RANDOM(elite_patterns)
        refined ← LLM_REFINE_PATTERN(pattern)
        new_patterns.append(refined)
    end for
    all_patterns ← MERGE_UNIQUE(patterns, new_patterns)
    LABEL_TRANSACTIONS(transactions, all_patterns)
    union_patterns ← DETECT_CO_OCCURRENCES(transactions)
    final_patterns ← TRUNCATE_TOP(all_patterns + union_patterns)
    for each pattern in final_patterns do
        suspicion_score ← avg_tax_amount
        audit_score = get fitness
    end for
    return final_patterns
end function
```

**Algorithm 5** Alternating Coevolution of Transactions and Audit Patterns

```
1: function EVOLUTION_ALTERNATING(num_generations)
2:     transactions, patterns ← INITIALIZE_POPULATION
3:     all_transactions ← [transactions]
4:     all_patterns ← [patterns]
5:     for generation ← 2 to num_generations do
6:         transactions ← EVOLVE_TRANSACTIONS(transactions, patterns)
7:         patterns ← EVOLVE_PATTERNS(patterns, transactions)
8:         transactions, patterns ← RELABEL_DATA(transactions, patterns)
9:         all_transactions.append(transactions)
10:        all_patterns.append(patterns)
11:    end for
12:    return all_patterns, all_transactions
13: end function
```

# B

# Appendix B: Prompts

## B.1 Prompts Used

### B.1.1 LLM-Tax Planner

#### System Message

You are a tax planner responsible for generating valid transactions within a network of entities and assets. Your goal is to propose valid financial transactions that align with the provided ownership rules, entity constraints, and asset guidelines. Transactions must follow the rules of ownership and should be creative.

#### Prompt for Generating Transactions

Your task is to propose one valid transaction within a network of entities and assets, and return it as a CSV file. Follow these guidelines:

**GUIDELINES**

1. A transaction involves two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean (True or False).
2. You MUST only use the exact entity names provided in the list below. Do not invent or modify entity names.

   Valid Entities: mr_jones, family_trust, new_co, mr_brown, jones_co
3. You MUST only use the exact asset names provided in the list below. Do not invent or modify asset names.

   Valid Assets: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, cash amounts up to $200
4. ENTITY1 gives ASSET1 to ENTITY2 in exchange for ASSET2:
   - ENTITY1 must own ASSET1 at the time of the transaction.

- ENTITY2 must own ASSET2 at the time of the transaction.

5. The following ownership details define which assets belong to which entities. You MUST only use these entities and their assets in your transactions:

{ownership_status}

6. mr_jones CANNOT trade partnership1 to new_co. This means the following setups are invalid:
- ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co
- ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1

7. An entity cannot transfer a partnership to another entity it is already in partnership with. These setups are invalid:
- ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = jones_co
- ENTITY1 = jones_co, ENTITY2 = mr_jones, ASSET2 = partnership1
- ENTITY1 = mr_jones, ASSET1 = partnership2, ENTITY2 = family_trust
- ENTITY1 = family_trust, ENTITY2 = mr_jones, ASSET2 = partnership2
- ENTITY1 = jones_co, ASSET1 = partnership3, ENTITY2 = new_co
- ENTITY1 = new_co, ENTITY2 = jones_co, ASSET2 = partnership3

8. If you use cash as an asset, put it as '$cash_amount'. The $cash_amount CANNOT exceed the amount of cash the entity currently owns.

**EXAMPLES**

{Examples}

**TASK**

Be creative! Choose diverse entities, assets, and transaction combinations while following the guidelines.

- Include a variety of entities and assets.
- Experiment with asset types: cash, promissory notes, hotel, partnerships.
- Use cash values between 0 and 200.
- Try both True and False for ELECTION754.

**FORMAT**

Return the transaction in the following CSV format. Include the column headers:

```
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

## Prompt for Mutating Transactions

Your task is to mutate a given transaction while adhering to the ownership rules and returning the mutated transaction as a CSV file. Follow these guidelines:

**GUIDELINES**

1. A transaction involves two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean (True or False).

2. You MUST only use the exact entity names provided below. Do not invent or modify entity names.
   Valid Entities: mr_jones, family_trust, new_co, mr_brown, jones_co
3. You MUST only use the exact asset names provided below. Do not invent or modify asset names.
   Valid Assets: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, cash amounts up to $200
4. ENTITY1 gives ASSET1 to ENTITY2 in exchange for ASSET2:
   - ENTITY1 must own ASSET1 at the time of the transaction.
   - ENTITY2 must own ASSET2 at the time of the transaction.
5. mr_jones CANNOT trade partnership1 to new_co. These setups are invalid:
   - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co
   - ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1
6. An entity cannot transfer a partnership to another entity it is already in partnership with. These setups are invalid:
   - mr_jones , jones_co (partnership1)
   - mr_jones , family_trust (partnership2)
   - jones_co , new_co (partnership3)
7. If using cash as an asset, write it as '$cash_amount'. The $cash_amount CANNOT exceed the amount of cash the entity currently owns.

**OWNERSHIP RULES — READ CAREFULLY**
Every transaction must follow the ownership information below:
`{ownership_status}`

- ENTITY1 must own ASSET1 at the time of the transaction.
- ENTITY2 must own ASSET2 at the time of the transaction.
- Do not assign an asset to an entity unless they are listed as its owner.
- If using cash (like "$100"), ensure the entity has at least that amount.
- Do not transfer partnerships between existing partners.
- Do not create transactions where ENTITY1 and ENTITY2 are the same.

**Common invalid cases (to avoid):**

- Giving away a promissory note not owned by the entity.
- Assigning hotel assets to incorrect owners.

- Trading partnerships with current partners.
- Using the same entity for both sides of the transaction.

**PREVIOUSLY MUTATED TRANSACTIONS**
Your mutated transaction must be different from each of the transactions below in at least one column (ENTITY1, ASSET1, ENTITY2, ASSET2, or ELECTION754). DO NOT REPEAT any of these:

```
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
{previous_mutations_csv}
```

**ORIGINAL TRANSACTION**
The current transaction is:

```
{transaction_df.to_csv(index=False)}
```

**MUTATION OPTIONS**
You may mutate the transaction in one of the following ways:

1. **Change the Asset Being Sold (ASSET1)**: Replace ASSET1 with another asset that ENTITY1 owns.
2. **Change the Asset Being Bought (ASSET2)**: Replace ASSET2 with another asset that ENTITY2 owns.
3. **Toggle the 754 Election Value**: Flip the ELECTION754 value (True → False or False → True).
4. **Change an Entity (ENTITY1 or ENTITY2)**: Replace one entity with another, ensuring they own the associated asset.

**TASK**
Choose one mutation type and apply it to the transaction. Your mutated transaction must:

- Be valid according to the ownership rules.
- Be different from all previously mutated transactions.
- Follow the structure and rules provided.
- BE CREATIVE.

**FORMAT**
Return the mutated transaction in the following CSV format, with no extra comments or text. Include the column headers:

```
‘‘‘csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

### B.1.2 LLM-Auditor

#### System Message

You are a tax auditor that analyzes transaction data in a tax network.
Your task is to identify general patterns and suspicious activity within the transaction data you observe.

#### Prompt for Generating Patterns

Dataset of transaction sequences and tax amounts:
`{formatted_data}`
**Task:**

- Identify `{max_patterns}` number of general patterns found in the individual transactions.

**Format:**
Return the identified patterns in a semicolon-separated file (similar to a CSV) as shown in the example, with `pattern_number` and `pattern_name` columns:
**semicolon-separated file**

```
pattern_number;pattern_name
1;pattern 1
1;pattern 2
3;pattern 3
```

Only return the semicolon-separated list, no comments or explanation needed.

#### Prompt for Labeling Patterns in Transaction Data

**Transaction sequence:**
`{transaction}`
**Identified patterns:**

```
\{categories_df.to_csv(index=False)\}
```

**Task:**

- Label the transaction with the matching `pattern_number(s)`.
- At least one pattern must be assigned.
- Return the result in a semicolon-separated format (similar to a CSV) with the following structure:

**semicolon-separated file**

```
identified_patterns
[pattern_number_1,pattern_number_2]
```

Ensure the `identified_patterns` column includes only `pattern_numbers` from the provided patterns and that they are formatted as a comma-separated list enclosed in square brackets.
Please just return the semicolon-separated file, no comments or explanations needed.

## Prompt for Mutating Patterns

**Your task is to MODIFY an existing transaction pattern that was previously identified in the dataset.**
Each row in the dataset represents a unique transaction sequence along with its corresponding tax amount. A transaction sequence consists of multiple individual transactions, enclosed in square brackets `[ ]`, with individual transactions separated by commas.
**Here are the transaction sequences and tax amounts in the dataset:**
`{formatted_data}`
**Existing pattern:**
`{existing_pattern}`
**Previously identified patterns:**

```
\{formatted_old_patterns\}
```

**Task:**

- MODIFY the existing pattern in a meaningful way so that it captures a more GENERAL, FREQUENT, and SUSPICIOUS pattern in the dataset.
- The new pattern must be meaningfully different from the existing pattern and from any of the previously identified patterns.
- Return the new pattern in a semicolon-separated file (similar to a CSV) as shown in the example:

**semicolon-separated file**

```
pattern_number;pattern_name
1;new pattern
```

Ensure the new pattern captures a GENERAL relationship between individual transactions.
Please return only the semicolon-separated file, no comments or explanations needed.

## B.2 Prompts Tested During Sensitivity Experiments

### B.2.1 LLM-Tax Planner

**Prompts for Generating Transactions:**

Minimal

Your task is to create one valid transaction between two entities who exchange assets. The transaction must include ENTITY1, ASSET1, ENTITY2, ASSET2, and a boolean ELECTION754 (True or False).

**GUIDELINES**

1. Use only the following entity names. Do not invent or modify names:
   Valid Entities: mr_jones, family_trust, new_co, mr_brown, jones_co
2. Use only the following asset names. Do not invent or modify names:
   Valid Assets: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, and cash amounts up to $200
3. ENTITY1 must currently own ASSET1, and ENTITY2 must currently own ASSET2. Use the ownership info below to determine this:
   `{ownership_status}`
4. Invalid transactions:
   - mr_jones cannot trade partnership1 to new_co (in either direction)
   - Entities cannot trade partnerships with entities they are partnered with:
     - mr_jones , jones_co (partnership1)
     - mr_jones , family_trust (partnership2)
     - jones_co , new_co (partnership3)
5. If using cash, format it as '$cash_amount'. The amount cannot exceed the entity's current cash.

**EXAMPLES**

`{Examples}`

**TASK**

Be creative! Choose different entities, assets, and cash values (between $0 and $200). Try both True and False for ELECTION754.

**FORMAT**

Return the transaction in the following CSV format, without additional comments or explanations. Include the column headers:

```
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Verbose Paragraph

Your task is to create one valid transaction involving two entities who exchange assets within a network. Each transaction must include two entities, called ENTITY1 and ENTITY2, two assets, called ASSET1 and ASSET2, and a boolean value named ELECTION754, which can be either True or False.
You must only use the exact entity names from the approved list: mr_jones, family_trust, new_co, mr_brown, and jones_co. Similarly, only use asset names from the valid list, without inventing or altering any: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, and cash amounts up to $200.
For a transaction to be valid, ENTITY1 must already own ASSET1, and ENTITY2 must already own ASSET2. You are provided with ownership details to determine what assets belong to which entities. You must use only this information when constructing your transaction.

`{ownership_status}`

There are several restrictions to keep in mind. mr_jones is not allowed to trade partnership1 to new_co. This restriction also applies in reverse, meaning that new_co cannot trade for partnership1 with mr_jones.
Additionally, no entity may transfer a partnership asset to another entity it is already partnered with. This restriction applies to the following specific cases:

- mr_jones and jones_co cannot trade partnership1
- mr_jones and family_trust cannot trade partnership2
- jones_co and new_co cannot trade partnership3

If you decide to use cash as one of the assets in the transaction, it should be written in the form '$cash_amount', where the amount cannot exceed the total cash currently owned by the entity.
Use the examples provided below as inspiration, but create a unique and valid transaction. Be creative in choosing different entities and asset types, such as promissory notes, hotels, partnerships, or cash. Try to use a variety of combinations, including different cash values between 0 and 200, and explore both True and False values for the ELECTION754 field.

`{Examples}`

Return your result as a CSV formatted response inside triple backticks. Do not include any additional text, line breaks, or explanations. Your response must contain exactly two lines: the header and a single transaction. The format must be exactly as shown below (no extra spaces or blank lines):

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Minimal Paragraph

Create one valid transaction involving two entities who exchange assets. Each transaction must include ENTITY1, ASSET1, ENTITY2, ASSET2, and a boolean ELECTION754 (True or False).
Use only these entity names: mr_jones, family_trust, new_co, mr_brown, jones_co. Use only these asset names: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, and cash (up to $200). ENTITY1 must own ASSET1, and ENTITY2 must own ASSET2. Use the following ownership info to determine this:

`{ownership_status}`

Invalid transactions: mr_jones cannot trade partnership1 with new_co (in either direction). No entity can trade a partnership with an entity it is already partnered with. This includes:

- mr_jones , jones_co (partnership1)
- mr_jones , family_trust (partnership2)
- jones_co , new_co (partnership3)

If using cash as an asset, write it as '$cash_amount' and do not exceed the entity's available cash. Use the examples below for guidance and create a new, creative transaction. Vary the entities, assets, cash (0–200), and ELECTION754 value (True or False).

`{Examples}`

Return your result as a CSV formatted response inside triple backticks. Do not include any additional text, line breaks, or explanations. Your response must contain exactly two lines: the header and a single transaction. The format must be exactly as shown below (no extra spaces or blank lines):

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Grammar

Your task is to propose one valid transaction within a network of entities and assets, and return it as a CSV file. Follow these rules carefully.

**GRAMMAR**

Use this grammar to structure all transactions syntactically:

`{grammar}`

**GUIDELINES**

These rules define what is semantically valid:

1. A transaction involves two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean (True or False).

2. ENTITY1 must own ASSET1 at the time of the transaction. ENTITY2 must own ASSET2 at the time of the transaction.
3. The valid entities and assets are already defined in the grammar — do not invent or modify them.
4. mr_jones CANNOT trade partnership1 to new_co. This means the following setups are invalid:
    - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co
    - ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1
5. An entity cannot transfer a partnership to another entity it is already in partnership with. Invalid combinations include:
    - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = jones_co
    - ENTITY1 = jones_co, ENTITY2 = mr_jones, ASSET2 = partnership1
    - ENTITY1 = mr_jones, ASSET1 = partnership2, ENTITY2 = family_trust
    - ENTITY1 = family_trust, ENTITY2 = mr_jones, ASSET2 = partnership2
    - ENTITY1 = jones_co, ASSET1 = partnership3, ENTITY2 = new_co
    - ENTITY1 = new_co, ENTITY2 = jones_co, ASSET2 = partnership3
6. If you use cash as an asset, put it as '$cash_amount'. The $cash_amount CANNOT exceed $200 for any entity.

**EXAMPLES**

`{Examples}`

**TASK**

**BE CREATIVE!** Choose diverse entities, assets, and transaction combinations while following the **GUIDELINES** section.

- Include a variety of entities and assets in your transaction.
- Experiment by using different types of assets like cash, promissory notes, hotels, and partnerships.
- Use a wide range of cash values, between 0 and 200.
- Experiment with the value of the ELECTION754 boolean.

**FORMAT**

Return the transaction in the following CSV format, without additional comments or explanations. Make sure to include the column names like this:

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Grammar + Explanation

Your task is to propose one valid transaction within a network of entities and assets, and return it as a CSV file. Follow these rules carefully.

**GRAMMAR AND EXPLANATION**

Use this grammar to structure all transactions syntactically:

```
{grammar}
```

Here's an explanation of the grammar:

- <**transactions**>: A transaction is a series of individual transactions, with each transaction being a combination of two entities, two assets, and an ELECTION754 boolean. The transaction sequence is separated by a space.
- <**transaction**>: Each individual transaction consists of the entities involved, the assets being exchanged, and the ELECTION754 boolean value.
- <**entity**>: Refers to valid participants in the transaction. In this case, valid entities are mr_jones, family_trust, new_co, mr_brown, and jones_co.
- <**asset**>: The assets involved can either be cash or a specific asset name. Cash amounts are capped at $200, and asset names refer to specific assets like promissory notes, partnerships, or hotels.
- <**cash**>: A cash value, represented as 200, can be used in transactions. However, cash amounts cannot exceed this value for any entity.
- <**asset_name**>: Refers to specific assets such as promissory notes or partnerships linked to particular entities (e.g., promissory1_mrjones, partnership1).
- <**election754**>: A boolean value that can either be True or False, used to define an additional aspect of the transaction.

**GUIDELINES**

These rules define what is semantically valid:

1. A transaction involves two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean (True or False).
2. ENTITY1 must own ASSET1 at the time of the transaction. ENTITY2 must own ASSET2 at the time of the transaction.
3. The valid entities and assets are already defined in the grammar — do not invent or modify them.
4. mr_jones CANNOT trade partnership1 to new_co. This means the following setups are invalid:
   - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co
   - ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1

5. An entity cannot transfer a partnership to another entity it is already in partnership with. Invalid combinations include:
    - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = jones_co
    - ENTITY1 = jones_co, ENTITY2 = mr_jones, ASSET2 = partnership1
    - ENTITY1 = mr_jones, ASSET1 = partnership2, ENTITY2 = family_trust
    - ENTITY1 = family_trust, ENTITY2 = mr_jones, ASSET2 = partnership2
    - ENTITY1 = jones_co, ASSET1 = partnership3, ENTITY2 = new_co
    - ENTITY1 = new_co, ENTITY2 = jones_co, ASSET2 = partnership3
6. If you use cash as an asset, put it as '$cash_amount'. The $cash_amount CANNOT exceed $200 for any entity.

**EXAMPLES**

`{Examples}`

**TASK**

**BE CREATIVE!** Choose diverse entities, assets, and transaction combinations while following the **GUIDELINES** section.

- Include a variety of entities and assets in your transaction.
- Experiment by using different types of assets like cash, promissory notes, hotels, and partnerships.
- Use a wide range of cash values, between 0 and 200.
- Experiment with the value of the ELECTION754 boolean.

**FORMAT**

Return the transaction in the following CSV format, without additional comments or explanations. Make sure to include the column names like this:

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Verbose + Grammar

Your task is to propose one valid transaction within a network of entities and assets, and return it as a CSV file. Follow these rules carefully.

**GRAMMAR**

Use this grammar to structure all transactions syntactically:

`{grammar}`

**GUIDELINES**

These rules define what is semantically valid:

1. A transaction involves two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean (True or False).
2. ENTITY1 must own ASSET1 at the time of the transaction.
   ENTITY2 must own ASSET2 at the time of the transaction.
3. The valid entities and assets are already defined in the grammar — do not invent or modify them.
4. mr_jones CANNOT trade partnership1 to new_co. This means the following setups are invalid:
   - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co
   - ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1
5. An entity cannot transfer a partnership to another entity it is already in partnership with. Invalid combinations include:
   - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = jones_co
   - ENTITY1 = jones_co, ENTITY2 = mr_jones, ASSET2 = partnership1
   - ENTITY1 = mr_jones, ASSET1 = partnership2, ENTITY2 = family_trust
   - ENTITY1 = family_trust, ENTITY2 = mr_jones, ASSET2 = partnership2
   - ENTITY1 = jones_co, ASSET1 = partnership3, ENTITY2 = new_co
   - ENTITY1 = new_co, ENTITY2 = jones_co, ASSET2 = partnership3
6. If you use cash as an asset, put it as '$cash_amount'. The $cash_amount CANNOT exceed $200 for any entity.
7. You MUST only use the exact entity names and asset names provided in the list below. Do not invent or modify entity or asset names.

   Valid Entities: mr_jones, family_trust, new_co, mr_brown, jones_co

   Valid Assets: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, cash amounts up to $200
8. The following ownership details define which assets belong to which entities. You MUST only use these entities and their assets in your transactions:

   `{ownership_status}`

**EXAMPLES**

`{Examples}`

**TASK**

**BE CREATIVE!** Choose diverse entities, assets, and transaction combinations while following the **GUIDELINES** section.

- Include a variety of entities and assets in your transaction.
- Experiment by using different types of assets like cash, promissory notes, hotels, and partnerships.
- Use a wide range of cash values, between 0 and 200.
- Experiment with the value of the ELECTION754 boolean.

**FORMAT**
Return the transaction in the following CSV format, without additional comments or explanations. Make sure to make the column names (ENTITY1, ASSET1, ENTITY2, ASSET2, ELECTION754) part of your response like this:

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Verbose + Grammar + Explanation

Your task is to propose one valid transaction within a network of entities and assets, and return it as a CSV file. Follow these rules carefully.
**GRAMMAR AND EXPLANATION**
Use this grammar to structure all transactions syntactically:
`{grammar}`
Here's an explanation of the grammar:

- <**transactions**>: A transaction is a series of individual transactions, with each transaction being a combination of two entities, two assets, and an ELECTION754 boolean. The transaction sequence is separated by a space.
- <**transaction**>: Each individual transaction consists of the entities involved, the assets being exchanged, and the ELECTION754 boolean value.
- <**entity**>: Refers to valid participants in the transaction. In this case, valid entities are mr_jones, family_trust, new_co, mr_brown, and jones_co.
- <**asset**>: The assets involved can either be cash or a specific asset name. Cash amounts are capped at $200, and asset names refer to specific assets like promissory notes, partnerships, or hotels.
- <**cash**>: A cash value, represented as 200, can be used in transactions. However, cash amounts cannot exceed this value for any entity.
- <**asset_name**>: Refers to specific assets such as promissory notes or partnerships linked to particular entities (e.g., promissory1_mrjones, partnership1).
- <**election754**>: A boolean value that can either be True or False, used to define an additional aspect of the transaction.

**GUIDELINES**

These rules define what is semantically valid:

1. A transaction involves two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean (True or False).
2. ENTITY1 must own ASSET1 at the time of the transaction.
   ENTITY2 must own ASSET2 at the time of the transaction.
3. The valid entities and assets are already defined in the grammar — do not invent or modify them.
4. mr_jones CANNOT trade partnership1 to new_co. This means the following setups are invalid:
   - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co
   - ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1
5. An entity cannot transfer a partnership to another entity it is already in partnership with. Invalid combinations include:
   - ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = jones_co
   - ENTITY1 = jones_co, ENTITY2 = mr_jones, ASSET2 = partnership1
   - ENTITY1 = mr_jones, ASSET1 = partnership2, ENTITY2 = family_trust
   - ENTITY1 = family_trust, ENTITY2 = mr_jones, ASSET2 = partnership2
   - ENTITY1 = jones_co, ASSET1 = partnership3, ENTITY2 = new_co
   - ENTITY1 = new_co, ENTITY2 = jones_co, ASSET2 = partnership3
6. If you use cash as an asset, put it as '$cash_amount'. The $cash_amount CANNOT exceed $200 for any entity.
7. You MUST only use the exact entity names and asset names provided in the list below. Do not invent or modify entity or asset names.

   Valid Entities: mr_jones, family_trust, new_co, mr_brown, jones_co

   Valid Assets: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, cash amounts up to $200
8. The following ownership details define which assets belong to which entities. You MUST only use these entities and their assets in your transactions:

   `{ownership_status}`

**EXAMPLES**

`{Examples}`

**TASK**

**BE CREATIVE!** Choose diverse entities, assets, and transaction combinations while following the **GUIDELINES** section.

- Include a variety of entities and assets in your transaction.
- Experiment by using different types of assets like cash, promissory notes, hotels, and partnerships.
- Use a wide range of cash values, between 0 and 200.
- Experiment with the value of the ELECTION754 boolean.

**FORMAT**
Return the transaction in the following CSV format, without additional comments or explanations. Make sure to make the column names (ENTITY1, ASSET1, ENTITY2, ASSET2, ELECTION754) part of your response like this:

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

## Prompts for Mutating Transactions

Minimal

Mutate the given transaction while following the ownership rules. Return the result as a CSV.
**GUIDELINES**

1. Each transaction includes: `ENTITY1`, `ASSET1`, `ENTITY2`, `ASSET2`, and `ELECTION754` (True/False).
2. Use only these entity names (do not modify or create new ones):
   `mr_jones, family_trust, new_co, mr_brown, jones_co`
3. Use only these asset names (do not modify or create new ones):
   `promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, cash up to $200`
4. `ENTITY1` gives `ASSET1` to `ENTITY2` in exchange for `ASSET2`:
   - `ENTITY1` must own `ASSET1`
   - `ENTITY2` must own `ASSET2`
5. `mr_jones` cannot trade `partnership1` with `new_co`. These setups are invalid:
   - `ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co`
   - `ENTITY1 = new_co, ASSET2 = partnership1, ENTITY2 = mr_jones`
6. Entities cannot trade a partnership with an entity they are already partnered with:
   - `mr_jones` ↔ `jones_co` (`partnership1`)

- mr_jones ↔ family_trust (partnership2)
- jones_co ↔ new_co (partnership3)

7. Use cash as $cash_amount (e.g., $150). The amount must not exceed what the entity currently owns.

**OWNERSHIP INFO**
Use the following ownership status to determine who owns which assets:

{ownership_status}

Keep all changes valid with respect to this ownership information.
**ORIGINAL TRANSACTION**

{transaction_df.to_csv(index=False)}

**ALLOWED MUTATIONS**
You may make one of the following changes:

1. Change ASSET1 (must be owned by ENTITY1)
2. Change ASSET2 (must be owned by ENTITY2)
3. Flip ELECTION754 (True ↔ False)
4. Change ENTITY1 or ENTITY2 (must still own the associated asset)

**FORMAT**
Return the mutated transaction in the following CSV format, without additional comments or explanations. Be sure to include the column names as shown below:

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Verbose Paragraph

Your task is to mutate a given transaction while following the ownership rules and returning the result as a CSV file.
Each transaction consists of two entities (ENTITY1 and ENTITY2), two assets (ASSET1 and ASSET2), and an ELECTION754 boolean value (True or False). You must only use the exact entity names provided: mr_jones, family_trust, new_co, mr_brown, and jones_co. Similarly, only use the exact asset names provided: promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown, and any cash amount up to $200. **Do**

**not invent or alter any entity or asset names.**
In each transaction, ENTITY1 gives ASSET1 to ENTITY2 in exchange for ASSET2. ENTITY1 must own ASSET1 at the time of the transaction, and ENTITY2 must own ASSET2.
Additionally, mr_jones is **not allowed** to trade partnership1 with new_co. This means transactions where ENTITY1 = mr_jones, ASSET1 = partnership1, ENTITY2 = new_co or where ENTITY1 = new_co, ENTITY2 = mr_jones, ASSET2 = partnership1 are invalid.
Furthermore, an entity cannot transfer a partnership asset to another entity it is already partnered with. This restriction applies to the following pairings:

- mr_jones and jones_co with partnership1
- mr_jones and family_trust with partnership2
- jones_co and new_co with partnership3

Any transactions involving these partnerships between the listed entities are invalid.
If you use cash as an asset, represent it in the format $cash_amount (e.g., $150). The amount **cannot exceed** the amount of cash the entity currently owns.

**OWNERSHIP STATUS**

Use the following ownership status to determine which entities own which assets:

{ownership_status}

All mutations must honor this ownership structure.

- If you change an **asset**, the entity involved must own the new asset.
- If you change an **entity**, that entity must still own the corresponding asset.
- If you change **both**, make sure the new entity still owns the new asset.

Any transaction that does not meet these ownership requirements is invalid.

**ORIGINAL TRANSACTION**

{transaction_df.to_csv(index=False)}

**ALLOWED MUTATIONS**

You may mutate the transaction in one of the following ways:

1. Change ASSET1, as long as the new asset is owned by ENTITY1
2. Change ASSET2, provided the new asset is owned by ENTITY2
3. Flip the ELECTION754 value from True to False or False to True
4. Change either ENTITY1 or ENTITY2, but only if the new entity still owns the same asset

**TASK**

Apply one of these mutation types to the transaction. Ensure the result is valid under the ownership rules and follows all of the above guidelines.

**FORMAT**

Return the mutated transaction as a CSV, using the exact format shown below. **Do not include any additional text, notes, or formatting.** Your response must contain exactly two lines: the header and a single transaction. The format must be exactly as shown below (no extra spaces or blank lines):

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

Minimal Paragraph

Mutate the following transaction by making a valid change and return it as a single-row CSV.
Each transaction includes two entities (`ENTITY1`, `ENTITY2`), two assets (`ASSET1`, `ASSET2`), and a boolean `ELECTION754` (True/False). Use only these exact entity names: `mr_jones, family_trust, new_co, mr_brown, jones_co`. Use only these asset names: `promissory1_mrjones, partnership1, partnership2, promissory1_jonesco, partnership3, promissory1_newco, hotel, promissory1_familytrust, promissory1_mrbrown`, or cash (up to $200, formatted as `$amount`).
`ENTITY1` gives `ASSET1` to `ENTITY2` in exchange for `ASSET2`. Both must already own the assets they are trading.
The following transactions are **not allowed**:

- `mr_jones` trading `partnership1` with `new_co`
- Any entity trading a partnership with an entity it already shares that partnership with (e.g., `mr_jones` ↔ `jones_co` with `partnership1`)

You must follow this ownership status when making changes:

`{ownership_status}`

**OWNERSHIP RULES**

- If you change an asset, the entity must own the new one.
- If you change an entity, it must still own the corresponding asset.
- If you change both, ownership must still be valid.

**ORIGINAL TRANSACTION**

`{transaction_df.to_csv(index=False)}`

**ALLOWED MUTATIONS**
You may make **one** of the following changes:

- Swap `ASSET1` (must be owned by `ENTITY1`)
- Swap `ASSET2` (must be owned by `ENTITY2`)
- Flip the `ELECTION754` value
- Change `ENTITY1` or `ENTITY2` (must still own the asset)

**FORMAT**
Return the mutated transaction as a CSV using the exact format shown below. **Do not include any additional text, notes, or formatting.** Your response must contain exactly two lines: the header and a single transaction. The format must be exactly as shown below (no extra spaces or blank lines):

```csv
ENTITY1,ASSET1,ENTITY2,ASSET2,ELECTION754
entity1,asset1,entity2,asset2,True/False
```

### B.2.2 LLM-Auditor

#### Prompt for Generating Patterns

Verbose

Each row in our dataset represents a unique transaction sequence along with that sequence's associated tax amount. Each transaction sequence consists of multiple individual transactions. A transaction sequence is enclosed in square brackets `[ ]`, and individual transactions within a sequence are separated by commas.
Here are the transaction sequences and corresponding tax amounts in the dataset:

`{formatted_data}`

Analyze the transaction sequences to identify:

1. GENERAL and potentially suspicious patterns in **individual transactions**.
2. Identify `max_patterns` number of patterns.
3. Return the identified patterns in a semicolon-separated file (similar to a CSV) as shown in the example, with `pattern_number` and `pattern_name` columns:

**semicolon-separated file**

```csv
pattern_number;pattern_name
1;pattern 1
1;pattern 2
3;pattern 3
```

Ensure the patterns capture the GENERAL relationships between individual transactions.
Please just return the semicolon-separated file, no comments or explanations needed.

### Prompt for Labeling Patterns

Verbose

Here is a single transaction sequence:

`{transaction}`

Here are the previously identified patterns and their associated pattern numbers:

`{categories_df.to_csv(index=False)}`

**Your task:**

1. Label this transaction sequence:
    - Go through all the identified patterns and check which patterns are present in the sequence.
    - If a pattern is recognized within the transaction sequence, include its `pattern_number` in the output.
    - You **MUST** label the transaction sequence with **AT LEAST ONE** pattern.
    - A transaction sequence can match multiple patterns, in which case, you should list **ALL** applicable patterns.
2. Return the result in a semicolon-separated format (similar to a CSV) with the following structure:

**semicolon-separated file**

```
‘‘‘csv
identified_patterns
[pattern_number_1,pattern_number_2]
```

Ensure the `identified`$_{p}$*atternscolumnincludesonly*

### Prompt for Making Patterns Clear

Verbose

**Context**
The dataset below contains a list of pattern numbers and their corresponding pattern names. These patterns are used to identify tax-related transaction behaviors in a complex dataset. Your task is to improve the clarity and precision of these pattern names so they are **easier to understand and**

**detect** when analyzing tax transactions.
**Data**
Here are the patterns:

`{formatted_data}`

**Instructions**
Please refine each pattern name by following these rules:

1. Ensure each name is written in **clear, straightforward English** so that it can be easily understood in the context of tax transactions.
2. Remove unnecessary words, redundant phrases, or technical jargon that does not aid comprehension.
3. Retain the original intent and meaning of each pattern.
4. **If a pattern name is already clear and precise, do not change it.**
5. **Keep the pattern_number exactly as it is. Do not change or reorder it.**

**Output**
Return the new pattern names in a semicolon-separated format exactly as shown below with the `pattern_number` and `pattern_name` columns:
**semicolon-separated file**

```csv
pattern_number;pattern_name
1;Refined Pattern 1
2;Refined Pattern 2
3;Refined Pattern 3
```

Return only the semicolon-separated file—no additional text, explanations, or comments.

Minimal

Here is a list of pattern numbers and their names. These patterns relate to tax transaction behavior.
**Patterns:**

`{formatted_data}`

**Your task** is to make the pattern names clearer and easier to understand. Keep the original meaning, but:

- Use simple, clear English
- Remove extra or confusing words

- Only change names that need improvement
- Do not change the pattern numbers or their order

**Output**
Return the new pattern names in a semicolon-separated format exactly as shown below with the `pattern_number` and `pattern_name` columns:
**semicolon-separated file**

```csv
pattern_number;pattern_name
1;Refined Pattern 1
2;Refined Pattern 2
3;Refined Pattern 3
```

Return only the semicolon-separated file—no additional text, explanations, or comments.

### Prompts for Mutating Patterns

Minimal

Modify the existing transaction pattern below to create a more **GENERAL**, **FREQUENT**, and **SUSPICIOUS** one.
**Dataset (transaction sequences + tax amounts):**

`{formatted_data}`

**Current pattern:**

`{existing_pattern}`

**Previous patterns:**

`{formatted_old_patterns}`

Return a new pattern that is clearly different from all above, in this format:
**semicolon-separated file**

```csv
pattern_number;pattern_name
1;new pattern
```

Only return the semicolon-separated file. No comments.

# C

# Appendix C: Additional Experiments

## C.1 T*: LLM-Tax Planner Experiments

### C.1.1 Mutation-Prompt Sensitivity Experiment

**Research Question and Experimental Design** We wanted to answer: *How sensitive is mutation performance to prompt phrasing and structure?* Using the same evolutionary settings as Table 4.2, we ran six trials for each mutation prompt. We used the metrics reported in Table 4.3. Note, in this experiment, we used a prior, outdated fitness function. But since this fitness function was held constant across trials, relative comparisons are still valid.

The mutation prompts compared were:

- **Verbose**: full, numbered prompt with ownership rules, examples, and option list (B.1.1).
- **Minimal**: concise version of verbose prompt (B.2.1).
- **Verbose Paragraph**: paragraph form of verbose prompt (B.2.1).
- **Minimal Paragraph**: minimal version of paragraph prompt (B.2.1).

**Results and Analysis** As shown in Table C.1, the fully structured **Mutate** prompt dominates on every accuracy metric: most iBOB discoveries, lowest errors, and best average and final-generation fitness. Simpler or paragraph variants reduce tokens and, in some cases, increase sequence diversity (lower embedding similarity, fewer duplicates) but pay for that flexibility with higher error rates and weaker fitness. The pattern mirrors our initialization

findings: rich, enumerated instructions act trade a small drop in diversity for substantially better correctness. Consequently, we adopt the original **Mutate** prompt in all subsequent work.

| **Exp.** | **Num iBOB** | **Num Errors** | **Avg. Fitness** | **Fitness Last Gen.** | **Cosine Sim.** | **Num Duplicates** | **Tokens** |
|---|---|---|---|---|---|---|---|
| Verbose | 25.83 (29.14) | 12.00 (7.69) | -101.43 (13.98) | -87.10 (39.87) | 0.95 (0.01) | 82.50 (2.17) | 238472 (9895) |
| Simple | 0.00 (0.00) | 29.50 (29.14) | -108.45 (14.06) | -105.33 (18.61) | 0.94 (0.02) | 83.83 (1.72) | 211816 (24286) |
| Verbose Paragraph | 3.67 (8.98) | 23.17 (18.94) | -127.54 (15.94) | -114.73 (42.21) | 0.96 (0.01) | 81.50 (2.95) | 219624 (17418) |
| Minimal Paragraph | 0.00 (0.00) | 37.33 (15.62) | -124.55 (19.98) | -120.43 (14.36) | 0.95 (0.01) | 80.50 (6.32) | 200183 (7258) |

Table C.1: LLM-Tax Planner, Mutation Prompt Sensitivity Experiment

**Note:** Each row shows the mean across 6 runs for a variation experiment. Columns include the number of iBOB transaction sequences discovered, total errors, average fitness across all generations, average fitness in the final generation, average embedding cosine similarity, number of duplicate transactions, and total tokens used. Numbers are averaged across six runs with standard deviations shown in paranthesis Best result is highlighted for each column.

## C.2 A*: LLM-Auditor Experiments

### C.2.1 Number of Patterns Experiment

**Research Question and Experimental Design** We investigate the factors affecting our LLM-Auditor evolution by asking: *How does the audit-pattern population size affect pattern quality during evolution?* Keeping the settings in Table 4.7, we initialized the LLM-Auditor with {2, 3, 4, 5, 6} patterns and ran six trials per condition; metric shown in Table 4.8.

**Results and Analysis** Table C.2 shows that fewer patterns inflate average pattern frequency by construction and are therefore uninformative. We instead examine cosine similarity: cosine similarity falls monotonically from two to four patterns (best=0.74) and then rises for five and six, indicating that four patterns span the transaction space most broadly without collapsing into redundancy. We therefore adopt **four audit patterns** as the default, balancing coverage with diversity in coevolution experiments.

| Experiment | Avg. Audit Frequency | Avg Audit Frequency Last Gen | Cosine Sim. | Total Tokens |
|---|---|---|---|---|
| 2 patterns | 0.97 (0.03) | 0.97 (0.04) | 0.77 (0.02) | 95669 (3281) |
| 3 patterns | 0.96 (0.04) | 0.96 (0.05) | 0.80 (0.03) | 102226 (3412) |
| 4 patterns | 0.95 (0.05) | 0.96 (0.04) | 0.74 (0.04) | 113155 (4789) |
| 5 patterns | 0.96 (0.04) | 0.99 (0.03) | 0.77 (0.04) | 129663 (4611) |
| 6 patterns | 0.95 (0.05) | 0.98 (0.03) | 0.78 (0.05) | 146368 (8739) |

Table C.2: LLM-Auditor, Number of Patterns Experiment

**Note:** This table presents results from varying the number of patterns. Metrics include audit frequency, audit frequency in final generation, output cosine similarity, and total token usage. Numbers are averaged across six runs with standard deviations shown in parenthesis.

### C.2.2 Co-occurring Patterns Experiment

**Research Question and Experimental Design** We next wonder how evolution is affected by pattern types, asking: *Does permitting pattern unions (co-occurrences) improve audit performance over evolution?* Using the settings from Experiment 3 (Table 4.7: ten fixed sequences, ten generations, four patterns, one labeling pass), we ran six trials in two conditions: **Union On** (unions of patterns can exist as patterns themselves) and **Union Off** (no unions of patterns).

**Results and Analysis** As reported in Table C.3, enabling unions yields a modest, though not statistically significant, lift in pattern frequency and a similar cosine similarity. Given this consistent edge and zero token cost, we retain the **Union On** setting for all coevolution experiments.

| Experiment | Avg. Audit Frequency | Avg Audit Frequency Last Gen | Cosine Sim | Total Tokens |
|---|---|---|---|---|
| No unions | 0.93 (0.04) | 0.95 (0.08) | 0.75 (0.03) | 120697 (5871) |
| Unions | 0.95 (0.05) | 0.96 (0.04) | 0.74 (0.04) | 113155 (4789) |

Table C.3: LLM-Auditor, Co-occurring Patterns Experiment

**Note:** This table compares the effect of using union patterns in prompt configurations. Metrics include audit frequency, last-generation audit frequency, cosine similarity, and token usage. Numbers are averaged across six runs with standard deviations shown in parenthesis.

## C.3 C*: Coevolution Experiments

### C.3.1 Examples of Fitness Calculations

| Transaction | Tax | Identified Patterns | Avg. Pattern Freq | Avg. Pattern Tax | Fit. 1 | Fit. 2 | Fit. 3 | Fit. 4 | Fit. 5 |
|---|---|---|---|---|---|---|---|---|---|
| Transaction seq. 1 | 80 | [1] | 1 | 40 | -81 | -180 | 39 | 3999 | 3839 |
| Transaction seq. 2 | 40 | [1, 3] | 0.833 | 30 | -40.833 | -123.3 | 29.17 | 2999.17 | 2919.12 |
| Transaction seq. 3 | 0 | [1, 2, 3] | 0.666 | 20 | -0.66 | -66.6 | 19.33 | 1999.33 | 1999.33 |

Table C.4: Example of Fitness Calculations on Transaction Sequences

**Note:** Avg. Pattern Freq represents the average frequency over all identified patterns within a transaction sequence. Each pattern has an Avg. Tax that calculates the average tax of all the transaction sequences it identifies. With this, Avg. Pattern Tax represents the Avg. Tax over all identified patterns within a transaction sequence.

| Patterns | Freq | Avg. Tax | Fit. 1 | Fit. 2 | Fit. 3 | Fit 4 | Fit 5 |
|---|---|---|---|---|---|---|---|
| Pattern 1 | 1 | 40 | 1 | 100 | -39 | -3999 | -3999 |
| Pattern 2 | 0.33 | 0 | 0.33 | 33 | 0.33 | 0.33 | 0.33 |
| Pattern 3 | 0.66 | 20 | 0.66 | 66 | -19.33 | -1999.34 | -1999.34 |

Table C.5: Example of Fitness Calculations on Patterns

**Note:** Freq represents the frequency with which a pattern identifies transaction sequences, represented as a proportion of all transaction sequences. Each pattern has an Avg. Tax that calculates the average tax of all the transaction sequences it identifies.

## C.3.2 Fitness Function Comparison Experiment

### Fitness 1

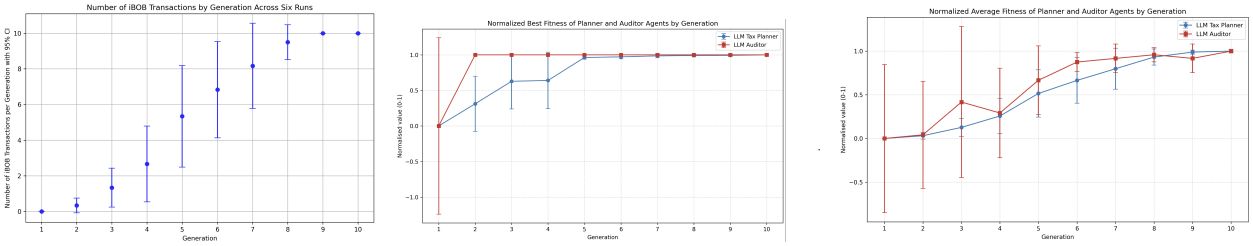


Figure C.1: (a) Number of iBOB Sequences per Generation Across Generations, (b) Best Fitness Across Generations, (c) Average Fitness Across Generations

### Fitness 2

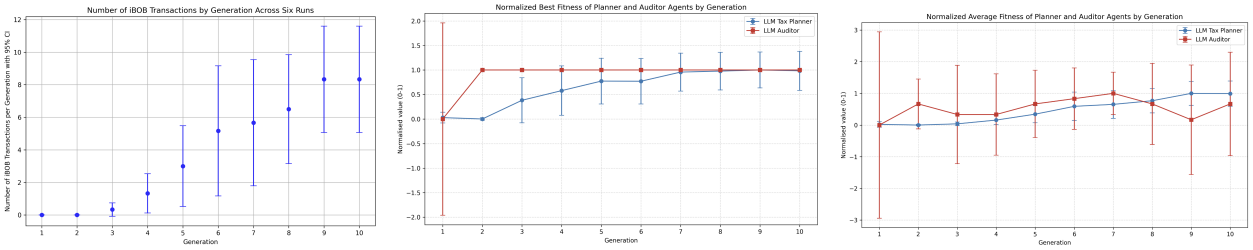


Figure C.2: (a) Number of iBOB Sequences per Generation Across Generations, (b) Best Fitness Across Generations, (c) Average Fitness Across Generations

### Fitness 3

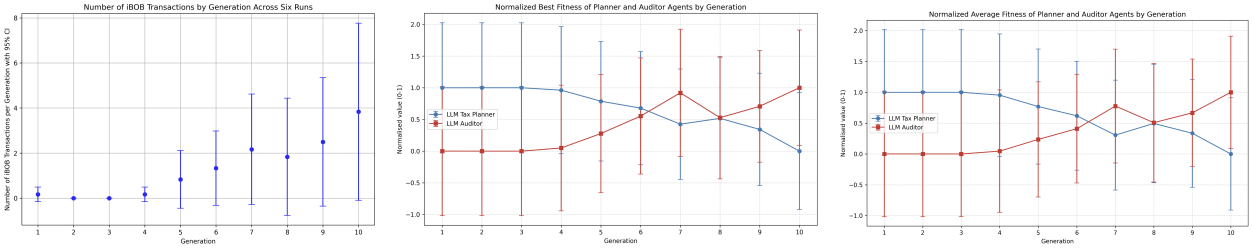


Figure C.3: (a) Number of iBOB Sequences per Generation Across Generations, (b) Best Fitness Across Generations, (c) Average Fitness Across Generations

## Fitness 4

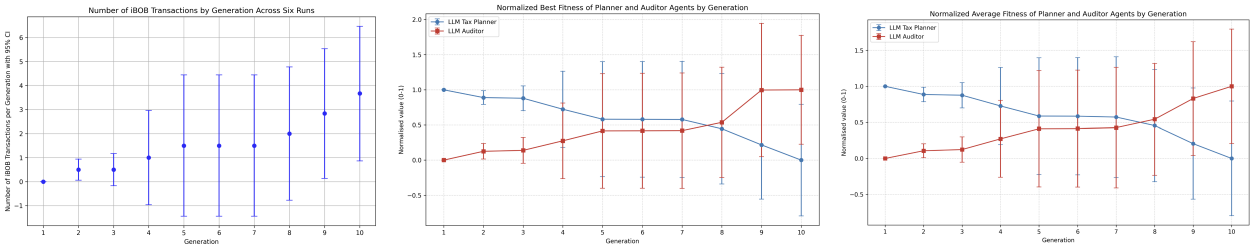


Figure C.4: (a) Number of iBOB Sequences per Generation Across Generations, (b) Best Fitness Across Generations, (c) Average Fitness Across Generations

## Fitness 5

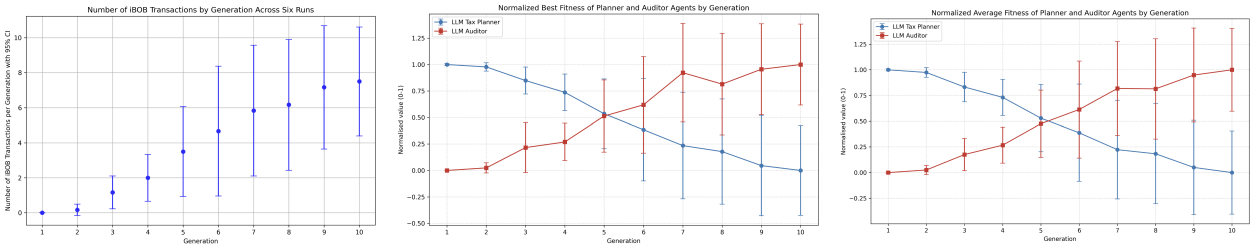


Figure C.5: (a) Number of iBOB Sequences per Generation Across Generations, (b) Best Fitness Across Generations, (c) Average Fitness Across Generations

### C.3.3 Coevolution vs. Benchmarks Experiment

#### Audit Evolution with Static Tax Planner

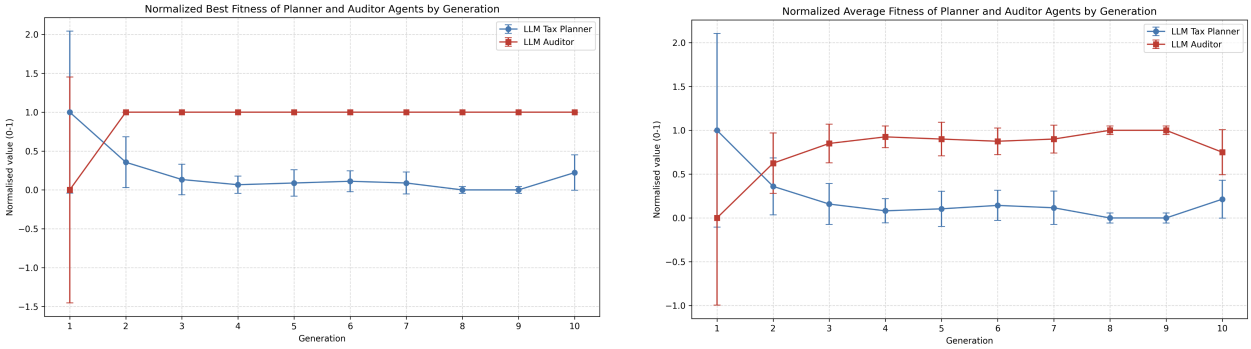


Figure C.6: (a) Best Fitness Across Generations, (b) Average Fitness Across Generations

**Note:** iBOB is never discovered, so we omit the iBOB graph.

#### Transaction Evolution with Static Auditor

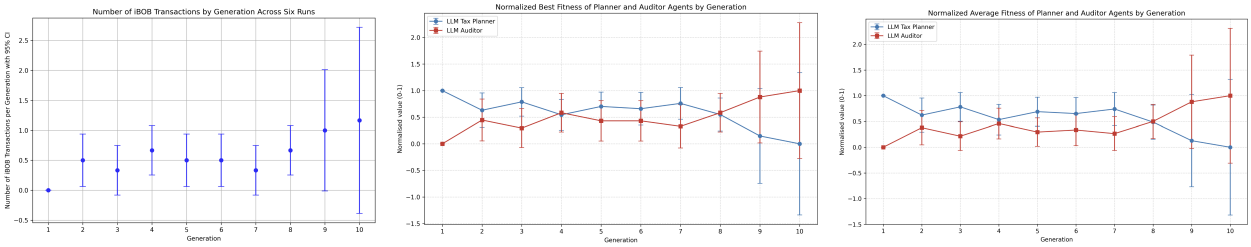


Figure C.7: (a) Number of iBOB Sequences per Generation Across Generations, (b) Best Fitness Across Generations, (c) Average Fitness Across Generations