SPACY

`zero-knowledge proofs (ZKP)`
`hardware wallets`
`SSI (self-sovereign identity)`

Foundational MPC & Threshold Cryptography

Goldreich-Micali-Wigderson (GMW) Protocol
"How to Play Any Mental Game" **(Goldreich et al., 1987)**
Shamir's Secret Sharing
"How to Share a Secret" **(Shamir, 1979)**
ZNP15
"Threshold Signatures for Blockchain Systems"

Wallet-Specific Security

BIP-32/39/44 Standards
"Bitcoin Improvement Proposals" **– Hierarchical deterministic (HD) wallets.**

"SoK: Blockchain Light Clients" (McCorry et al., 2019) – Wallet security analysis.

Privacy & Authentication

"Zerocash" (Sasson et al., 2014) – Privacy-preserving transactions.

"uPort: Decentralized Identity" (Lundkvist et al., 2016) – SSI for wallets.

Hardware Wallets

"Trezor & Ledger Security Analysis" (TriumphCrypto, 2020) – Side-channel attacks.

MIT Media Lab (ZNP15) – Pioneered threshold crypto for wallets.

Ethereum Foundation – Research on MPC and smart contract wallets (e.g., ERC-4337).

Stanford (Dan Boneh) – Work on threshold signatures and ZKPs.

University College London – Contributions to blockchain privacy (McCorry et al.).

ERC-4337 (Account Abstraction) – Ethereum's smart contract wallets.

FIDO2 – Hardware-backed authentication for wallets.

W3C DID – Decentralized identity standards.

CoinDesk Research – Industry adoption reports.

NIST IR 8401 – Standards for crypto key management.

Usenix Security

Analysis of Outputs (that should be fixed)

- ➢ Tone shifts from technical explanations to casual languages
- ➢ Repetitive formats before responding
- ➢ Shifts of point-of-view in between first-person and impersonal
- ➢ Misrepresentations and incorrect definitions
- ➢ Circular definitions in explaining terms
- ➢ Reponses frequently trail off mid-sentence
- ➢ Lacks clear thesis statements or conclusions
- ➢ Introduce irrelevant details
- ➢ Grammatically incomplete outputs
- ➢ Hallucinations. Fabricated references.
- ➢ False timeline.
- ➢ Fails to distinguish and in fact confuses concepts
- ➢ Ignore the questions' requirements (define, summarize…)
- ➢ Copies prompts and adds erratic spacing.
- ➢ Overuses casual phrases and emojis
- ➢ Rambling, unstructured thoughts
- ➢ Unnatural breaks because of overuse of EOS

How to fix these errors and make the model more professional and factually reliable

- Implement fact-checking against knowledge bases
- Add post-generation validation for completeness
- Abides by max_length
- Proper stopping.
- Curated Q&A pairs.
- Needs `temperature=0.5`, `stopping_criteria`, and `no_repeat_ngram_size`.

Solutions:

1. Data Curation & Preprocessing

Problem: Hallucinations, factual errors, tone shifts

Solution:

Key Actions:

- Source domain-specific datasets (e.g., crypto/tech for wallet questions)
- Remove informal/personalized examples ("I think...", "In my experience...")
- Add EOS tokens only after verified complete sentences

---

2. Prompt Engineering

Problem: Repetition, erratic spacing, ignored instructions

2. Critical Grammar Guards

Key Adjustments:

- **Transitional phrase validation replaces bullet-point checks**
- **Focus on pronoun continuity and causal connectors**

---

3. Optimization for Technical Accuracy

---

4. Anti-Hallucination Measures

```
flow_verification_rules = [

    {"name": "Example Relevance",

     "pattern": r"(For example|e\.g\.)\s+([^\.]+\.)",

     "action": "validate_against_knowledge_graph"},
```

```
    {"name": "Definition Consistency",
     "pattern": r"(is defined as|refers to)\s+([^\.]{15,50}\.)",
     "action": "crosscheck_with_glossary"}
]
```

```
banlist = [
    "In summary", "To conclude",  # Forces organic conclusions
    "The following points",  # Prevents list-thinking
    "On one hand"  # Avoids artificial dichotomy setups
]
```

Key Actions:

- Enforce instructional prompts (e.g., "Define X in one sentence.")
- Use few-shot learning with 3-5 properly formatted examples
- Add negative examples (e.g., "Do NOT mention personal opinions")

---

3. Generation Parameter Optimization

Problem: Incomplete sentences, rambling, casual tone

Key Actions:

- Lower temperature for factual accuracy
- Enable `no_repeat_ngram_size` to avoid redundancy
- Implement dual stopping criteria (length + punctuation)

---

4. Custom Stopping Criteria

Problem: Mid-sentence breaks, EOS overuse

Key Actions:

- Force stops only at punctuation or line breaks
- Check last 10 tokens (not just 5) for better context

---

5. Post-Processing Pipeline

Problem: Spacing issues, prompt repetition, informality

Solution:

```
def clean_response(response, prompt):
    # Remove prompt repetition
    response = response.replace(prompt, "").strip()

    # Fix spacing
    response = " ".join(response.split())

    # Remove casual phrases
    casual_phrases = ["well,", "you know", "I think", ":)", "let me explain"]
    for phrase in casual_phrases:
        response = response.replace(phrase, "")

    # Ensure sentence completion
    if not any(response.endswith(p) for p in [".", "?", "!"]):
        response = response.rsplit(".", 1)[0] + "." if "." in response else response

    return response
```

---

6. Fact-Checking Layer

Problem: Hallucinations, incorrect timelines

Solution:

```
from fact_checker import FactChecker  # Hypothetical module

fact_checker = FactChecker(domain="cryptocurrency")

def validate_response(response):
    claims = extract_claims(response)  # NLP to isolate factual statements
    for claim in claims:
        if not fact_checker.verify(claim):
            return False
    return True
```

Key Actions:

- Pre-train a classifier to flag unverifiable claims
- Use knowledge graphs (e.g., Wikidata) for real-time validation

---

7. Output Structure Enforcement

Problem: Lacking thesis/conclusions, digressions

Solution:

```
structure_rules = {
    "define": "Concept: <term>\nDefinition: <2-3 sentences>\nExample: <1>",
    "compare": "Differences:\n1. <A> vs <B>\n2. <A> vs <B>\nConclusion: <1 sentence>"
}

def enforce_structure(response, prompt_type):
    template = structure_rules.get(prompt_type, "Freeform")
    # Use LLM to reformat response into template
    return model.format_response(template, raw_response=response)
```

---

8. Human-in-the-Loop Verification

Problem: Residual errors after automated fixes

Solution:

```
def human_review(batch):
    for response in batch:
        if confidence_score(response) < 0.8:  # ML confidence threshold
            send_for_human_review(response)
```

Implementation:

- Deploy on platforms like Label Studio for rapid review
- Focus on borderline cases (0.6-0.8 confidence scores)

---

9. Continuous Monitoring

Problem: Drift over time

Solution:

```
python
Copy

Download
monitor = QualityMonitor(
    metrics=["fact_accuracy", "readability", "relevance"],
    alert_thresholds=[0.9, 0.85, 0.95]
)

for response in production_outputs:
    monitor.log(response)
    if monitor.alert_triggered():
        retrain_model()
```

---

10. Testing Protocol

Problem: Undetected edge cases

Solution:

```
test_cases = {
    "What is PoW?": {
        "required_terms": ["consensus", "mining", "difficulty"],
        "banned_terms": ["book", "article", "PoS"]
    }
}

def run_tests():
    for prompt, rules in test_cases.items():
        response = generate(prompt)
        assert all(term in response for term in rules["required_terms"])
        assert not any(term in response for term in rules["banned_terms"])
```

---

11. Documentation Standards

For maintainability:

```
# Response Quality Guidelines
1. **Accuracy**: All facts must be verifiable via `FactChecker`
2. **Structure**: Use templates from `structure_rules.py`
3. **Style**: Formal, third-person, no colloquialisms
4. **Length**: 50-100 tokens, complete sentences only
```

Dataset Enhancement Strategy

1. Targeted Q&A Expansion (5,000-50,000 Pairs)

Priority Areas:

python

Copy

Download

```
categories = {
    "Wallet Security": [
        ("Compare Shamir Backup vs. BIP-39 mnemonics",
         "Shamir Backup splits secrets into shares (e.g., 3-of-5), while BIP-39 uses a single 12-24 word phrase. Shamir offers better loss protection but requires compatible wallets like Trezor Model T."),
        ("How do hardware wallets prevent MITM attacks?",
         "They verify transaction details on-device displays and use secure elements to isolate private keys from connected devices.")
    ],
    "Transaction Mechanics": [
        ("Why do Ethereum gas fees spike during NFT mints?",
         "Block space becomes scarce as users bid higher fees for priority. EIP-1559's base fee helps but can't eliminate demand surges.")
    ]
}
```

2. Phrase Diversity Injection

Techniques:

1. **Paraphrasing**: Use GPT-4 to generate 5-10 variants per question
2. python
3. Copy
4. Download
5. `prompt = "Generate 8 differently phrased questions about: 'How do I recover a lost hardware wallet?'"`

6. **Common Misphrasings**: Include user-like errors
    - "How recover lost ledger?"
    - "My Trezor is broke how get coins?"

3. Anti-Hallucination Training

Data Augmentation:

python

Copy

Download

```
{
    "question": "What is BitFurenceWalletChain?",
    "answer": "[REJECT] This appears to be a hallucinated term. Valid wallet types include: Ledger, Trezor, MetaMask...",
    "category": "Error Correction"
}
```

4. Structural Enforcement Samples

Template-Based Examples:

markdown

Copy

Download

```
# For "compare" prompts:
Differences:
1. **Hot vs Cold Wallets**:
   - Hot: Connected to internet (MetaMask)
```

```
   - Cold: Offline storage (Ledger)

2. **Security**:

   - Hot vulnerable to phishing

   - Cold immune to remote attacks

Conclusion: Cold wallets better for long-term storage.
```

Implementation Roadmap

1. **Phase 1 (500-5,000 pairs)**
    - Scrape & clean data from:
        - Wallet docs (Trezor, Ledger GitHub wikis)
        - Stack Exchange (Bitcoin/ETHerum tags)
        - Certik security audit reports
2. **Phase 2 (5,000-20,000 pairs)**
    - Generate synthetic Q&A using:
    - python
    - Copy
    - Download

```
llm.generate(

    template="Explain [concept] to a [beginner|developer] in [1 sentence|3 bullet points]",

    concepts=["multisig", "HD wallets", "gas optimization"]
```

    -)
3. **Phase 3 (20,000-50,000 pairs)**
    - Implement active learning:
        - Flag low-confidence model predictions for human review

- Prioritize edge cases (e.g., "How to recover funds sent to wrong network?")

Evaluation Framework

python

Copy

Download

```python
def test_improvements():
    test_cases = [
        {
            "prompt": "Compare hot and cold wallets in 3 bullet points",
            "metrics": [
                "Correct term usage (score 0-5)",
                "Structural compliance (score 0-3)",
                "Hallucination detection (score 0-2)"
            ]
        }
    ]
    # Implement before/after testing
```

Expected Outcome: 30-50% improvement in accuracy and structure adherence after implementing the enhanced dataset.