ITEM #266 Supporting Document:
Passive-Code-Hits + Tail-Length Structural Decoding over Shared Random Bit Streams
（基于共享随机比特流的被动触动码 + 尾长结构解码）

English Version

1. Motivation

This ITEM documents a **structural decoding algorithm** inspired by an exploration of whether quantum-correlated measurements could support ultra-low-bandwidth communication.
During discussion, a critical clarification emerged:
The receiver’s measurement sequence must be treated as a **shared random bit stream**, not as “signal + noise”.
This distinction prevents conceptual errors and allows the algorithm to be studied as a **pure structural decoding system**.
The algorithm therefore belongs to:

Shared randomness → structural decoding → language projection
not to:

quantum coherent communication
This ITEM focuses on the decoding architecture itself.

2. System Abstraction

We assume both sides share a synchronized random bit stream:

y1, y2, y3, y4, ...
This stream may originate from:

- quantum-correlated measurements
- synchronized hardware randomness
- shared entropy sources
- simulation

The source is irrelevant to the decoding algorithm.

3. Core Idea

Instead of reading the bit stream sequentially, we interpret it using:

- Head-code reading
- Tail-length skipping
- CallingGraph pruning

This converts a random bit stream into structured token sequences.

4. Encoding Table (Word Encoding Table)

A shared table defines:

(head code) → (candidate words, tail-length)

Example:

Head	Candidate Words	Tail-Length
00	奥, 嗯	8
01	博	3
10	特	15
11	你	7

The table is optimized to:

- minimize collisions
- reduce branching factor
- maximize structural separability

5. Decoding Algorithm

Step-by-step procedure

Receiver performs:

```
pos = 0
while not END:
    head = read 2 bits at pos
    candidates = table.lookup(head)

    for each candidate:
```

```
        next_pos = pos + 2 + tail_length(head)
        expand permutation tree

    prune using CallingGraph
    pos = next frontier
```
This produces a **Permutation Tree constrained by language structure**.

6. Role of Tail-Length

Tail-length does NOT skip channel noise.
Instead, it:

skips portions of the random bit stream
and reduces search branching.
This is a structural compression mechanism.

7. Complexity Behavior

Let:

- L = average tail length
- H = head bits
- B = CallingGraph branching
- T = token count

Then:

bits per token ≈ H + L
tree size ≈ B^T
T ≈ N / (H+L)
Tail-length increases step size and reduces tree depth.

8. CallingGraph Pruning

CallingGraph acts as:

finite-state grammar constraint
and eliminates most branches early.
Typical branching after pruning:

1–3

9. Repetition + Vote

Multiple decoding runs can be aggregated:

run decoding multiple times
collect candidate sentences
vote by frequency or structural score
This stabilizes the projection from randomness to language.

10. What This Algorithm Is (and Is Not)

It IS

- Structural decoding
- Grammar-guided search
- Permutation-tree traversal
- DBM-style constrained interpretation
- Random-stream language projection

It is NOT

- quantum communication
- signaling through entanglement
- information transmission mechanism

The receiver sequence must be treated as:

shared randomness
not:

signal + noise

11. Relationship to DBM

This algorithm resembles:

- BTP permutation traversal
- IR interpretation layers

- CCC extraction from stochastic streams
- grammar-constrained decoding

It can be viewed as:

Random IR → Structural Interpreter → CCC Projection

12. Implementation Notes

A minimal implementation requires:

- EncodingTable
- BitStreamReader
- PermutationTreeNode
- CallingGraphValidator
- VoteAggregator

No quantum hardware is required.

13. Educational Note

A common misunderstanding is to assume:

tail-length skips noise
Correct interpretation:

tail-length skips random stream segments
This distinction is essential.