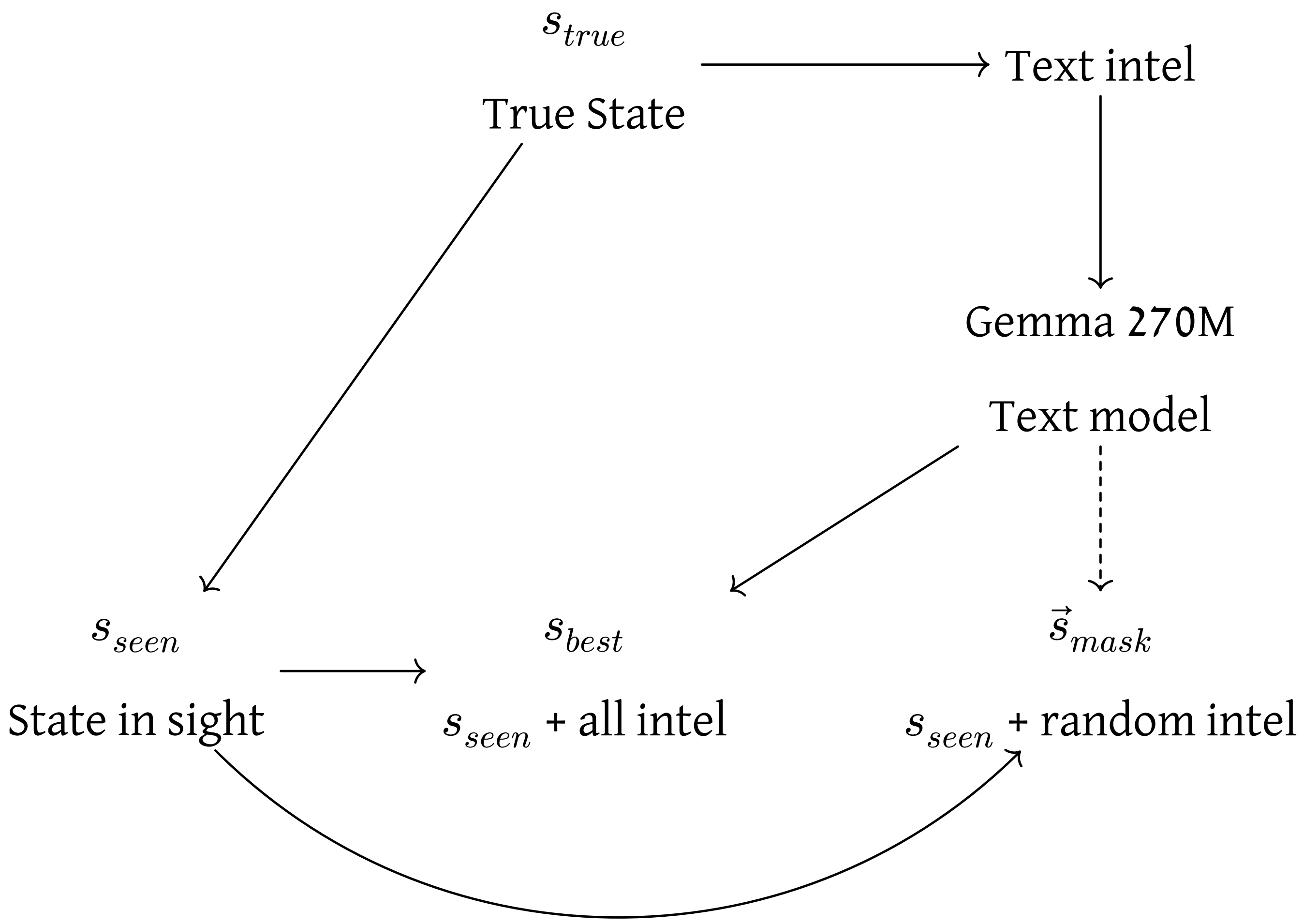
A Semantic Signal Processing Framework

Noah Syrkis

September 18, 2025


s_{true}
True State
Text intel
Gemma 270M
Text model
s_{seen}
State in sight
s_{best}
s_{seen} + all intel
$\vec{s}_{mask}$
s_{seen} + random intel

- New RNG dimension for each of k sim sims
- Quantization of state (Figure 2) during play
- Today: improve masking (uniform in space)

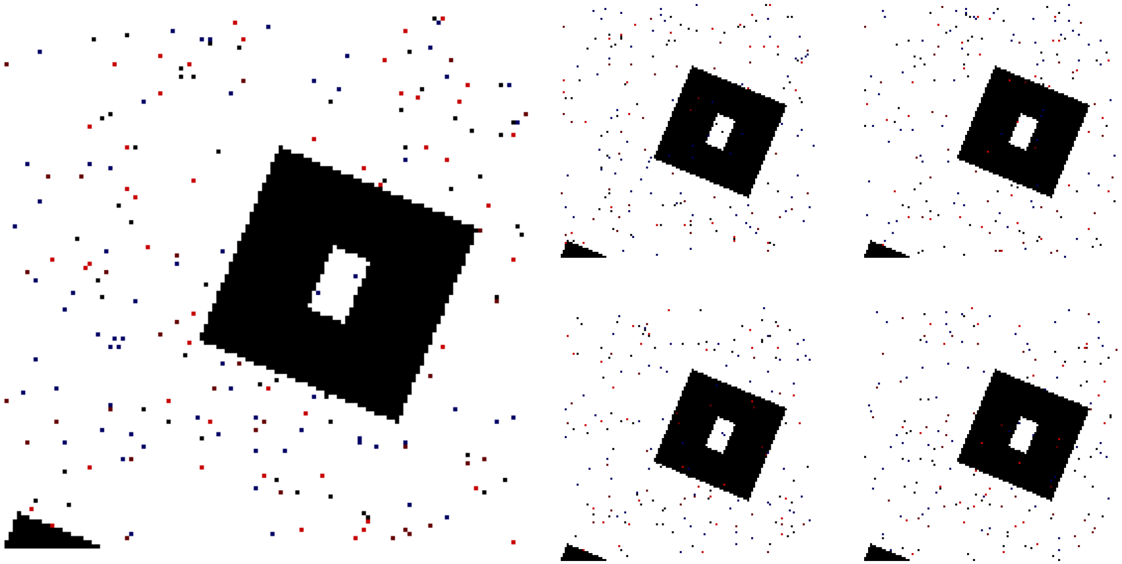


Figure 2: A single base simulation $\vec{s}_t$ (left) and four imagined simulations $\vec{\vec{s}}_t$(right)

1 | Introduction

Nebellum is a semantic signal processing framework. It leverages a multimodal large language model, Gemma, to decode pieces of intel, and a vectorized war game, Parabellum, to assign importance to those pieces.

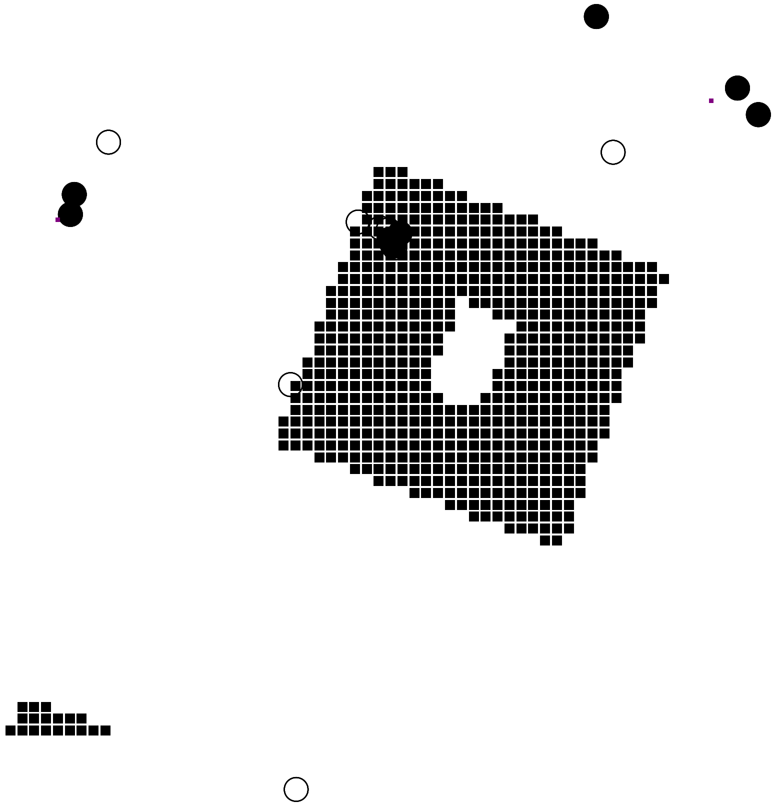


Figure 3: Parabellum simulation of Colosseo Quadrato in Rome

1.1 | Notation

This is simulation heavy project. We have true game states, various kinds of estimated game states, as well as simulated game states

s_t	A true game state at time t
s'_t	Simulated game state based on s_t
$\hat{s}_t$	An intel subset based estimate of s_t
$\hat{s}'_t$	Simulated game state based on s'_t
$\overline{s}_t$	Estimte of s_t based on all intel
$\overline{s}'_t$	Simulated state based on $\overline{s}_t$

2 | Architecture

- Low level behavior is controlled by assigning behavior trees b and target positions
- Behavior trees map observations (info on units in sight range) to actions (move or shoot vector)
- Unit behavior (and target) is assigned by evaluating plan p at time t
- Plan evaluation happens m (evenly spaced) times throughout an n step episode

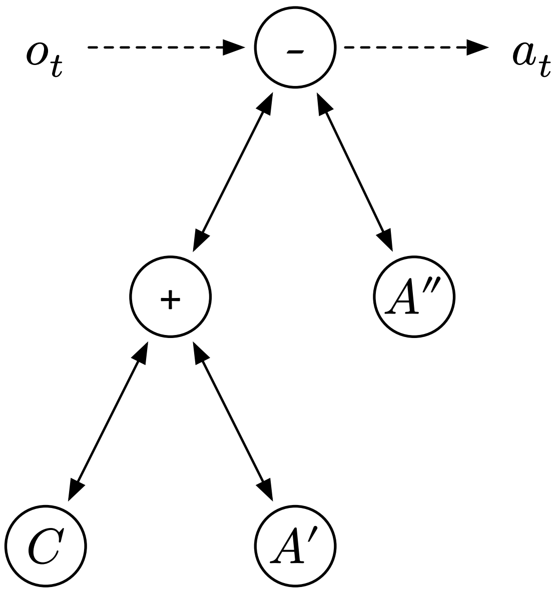


Figure 4: Behavior tree receiving a unit observation o_t and returning a unit action a_t

- Leafs are conditions (C) or actions (A)
- Non-leafs are sequnces (+) or fallbacks (-)
- They require all (+) or one (-) child success
- In Figure 4, if C succeds A' runs else A'' runs
- "if enemy is in reach shoot enemy else move up"

2 | Architecture

- Plan evaluation maps state s_t to behavior b_t
- Figure 5 shows a graph with nodes that contain unit group g, target t and behavior b
- Targets can be positional (g shall go near t) ...
- ... or adversarial (g shall kill enemies near t)

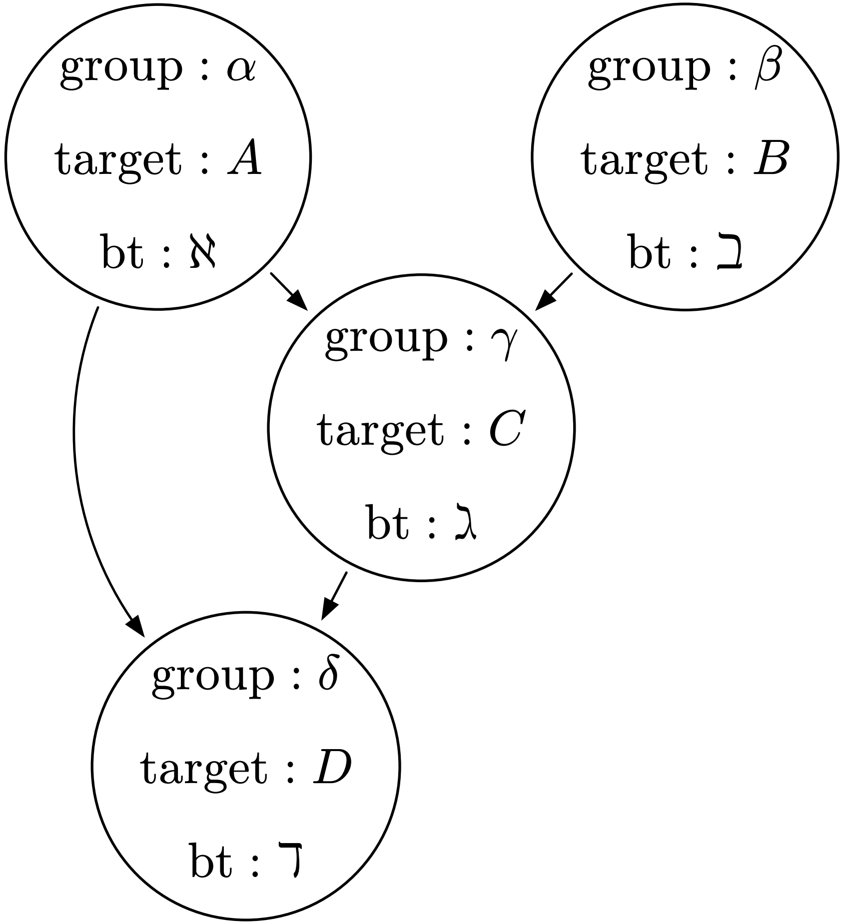


Figure 5: Example of plan data structure

2 | Architecture

$$
\begin{tikzcd}
 & i_t \arrow[dr] & & \\
s_t \arrow[ur] \arrow[dr] & & \hat{s}_t \arrow[r, squiggly] & \begin{pmatrix} \hat{s}'_{(t+m)1} \\ \hat{s}'_{(t+m)2} \\ \vdots \\ \hat{s}'_{(t+m)k} \end{pmatrix} \\
 & o_t \arrow[ur] & &
\end{tikzcd}
$$

Figure 6: At time t we have state s_t, intel i_t and observation o_t. These are combined into $\hat{s}_t$, the basis for k different m step trajectories

Each of the k plan evaluations consists of:

1. Analysis: estimating the state $\hat{s}_t$ by combining unit observation o_t with intel i_t
2. Simulation: setting k imagined simulations in motion based on $\hat{s}_t$

2.1 | Analysis

- Intel i_t is generated on all units in s_t encoding position and health in natural language
- Each piece of intel is fed to Gemma for analysis and combination into state estimate $\hat{s}_t$
- For each team, k intel subsets are made from a random masking of enemy units out of sight
- For each intel subset $i_{tj} : j \in [1, k]$ state $\hat{s}_{tj}$ is made by masking info in $\hat{s}_t$ from intel not in i_{tj}

`{pos}!! look hurt, maybe {hp} health?'`
$\vdots$
`a stranger at {pos}. health {hp}?`

$\downarrow$

(x, y)	01
$\vdots$	$\vdots$
(x, y)	01

$\downarrow$

$$\begin{pmatrix} \hat{s}_{t1} & \hat{s}_{t2} & \dots & \hat{s}_{tk} \end{pmatrix}$$

2.2 | Simulation

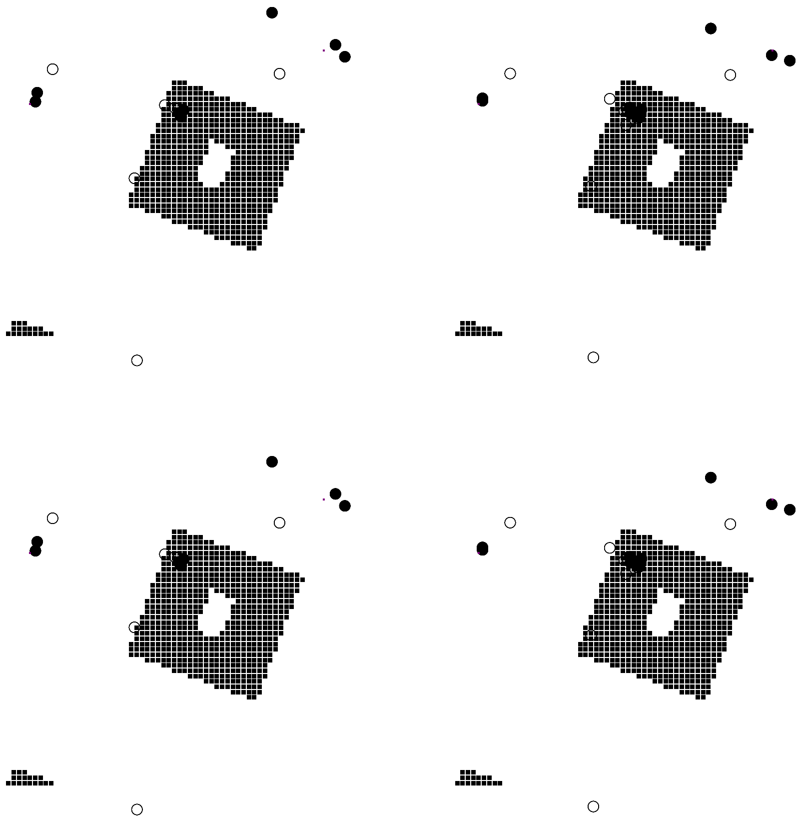


Figure 7: Simulated futures based on would be behaviors based on different pieces of intel

- The plan p is evaluated for each of the k states $\hat{s}_{tj} : j \in [1, k]$ yielding the k behaviors b_j
- From s_t or $\hat{s}_{tj}$, k trajectories (one for each behavior b_j) of length $\lfloor \frac{n}{m} \rceil$ are run and recorded

2.2 | Simulation

- Using s_t as simulation basis we can assign importance to different aspects of the intel i_{tj}
- Using $\hat{s}_{tj}$ we can assign importance to different aspects of the current state s_t

3 | Evaluation

- As expected the delta starts small
- Divergence occus under some conditions
- For a subset of these, it converges again

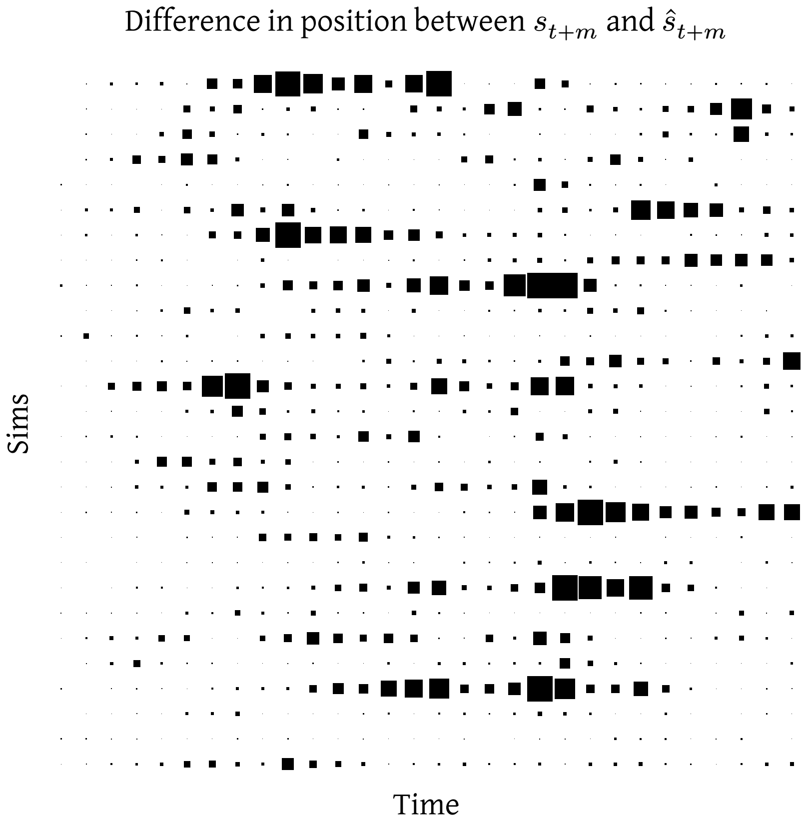


Figure 8: Diverence between true state s_t and simulated states. Simulations are based on s_t (not $\hat{s}_t$. They thus difer in behavior and random seed.

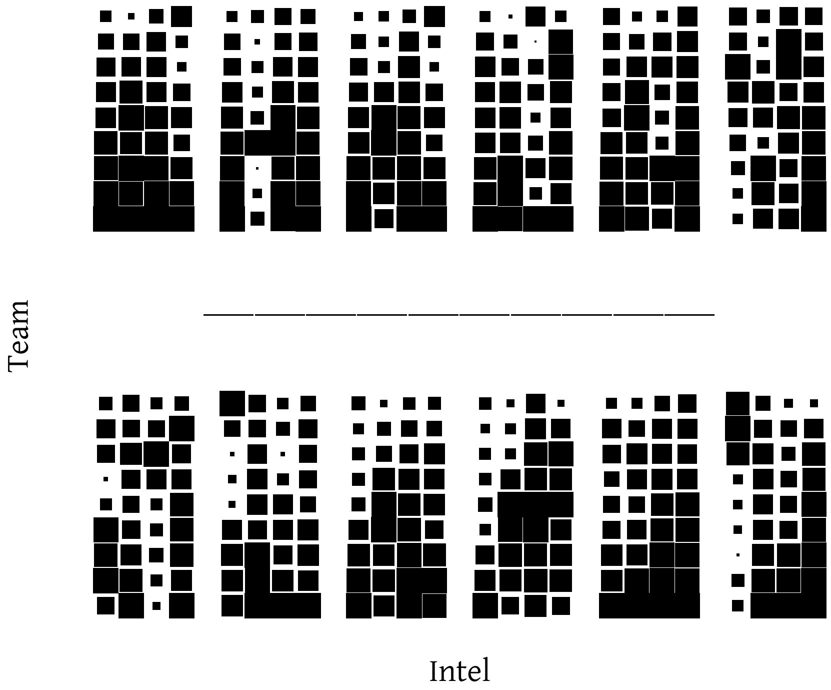


Figure 10: Linear regression β values of importance of different pieces of intel

- In subplots rows is epoch and cols is time
- We see that there is a pattern
- Sometimes somethings predict the future ...
- ... other times other things predict the future

4 | Future

- [x] Compute importance score
- [ ] Highlight which intel is important
- [ ] Add importance overlay to location on top of map (nice UI)
- [ ] Multiple trajectories s' per intel subset (for robustness)
- [ ] Hide true enemy plan, infering it from o_t and i_t
- [ ] Non visible enemy units could be sampled uniformly?

References

A | Stochastic Signal Processing

We denote the weights of a model as θ. The gradient of θ with respect to our loss function at time t we denote $g(t)$. As we train the model, $g(t)$ varies, going up and down. This can be thought of as a stocastic signal. We can represent this signal with a Fourier basis. GrokFast posits that the slow varying frequencies contribute to grokking. Higer frequencies are then muted, and grokking is indeed accelerated.

B | Discrete Fourier Transform

Function can be expressed as a linear combination of cosine and sine waves. A similar thing can be done for data / vectors.

C | Singular Value Decomposition

An $n \times m$ matrix M can be represented as a $U\Sigma V^*$, where U is an $m \times m$ complex unitary matrix, Σ a rectangular $m \times n$ diagonal matrix (padded with zeros), and V an $n \times n$ complex unitary matrix. Multiplying by M can thus be viewed as first rotating in the m-space with U, then scaling by Σ and then rotating by V in the n-space.