Il progetto mette insieme due filoni complementari: da un lato la compilazione della conoscenza (knowledge compilation) per ottenere un conteggio esatto dei modelli (#SAT) tramite la forma sd-DNNF, dall'altro un approccio stocastico che stima #SAT in modo approssimato. Nel mezzo ci sono utilities per rappresentare formule, generare assegnazioni e visualizzare i risultati.

Si parte dalla pipeline logica. Con to_nnf si porta la formula in NNF (Negation Normal Form), cioè una forma in cui le negazioni compaiono solo davanti ai letterali. La funzione NNF2DNNF mira poi alla DNNF (Decomposable NNF): la decomponibilità richiede che in ogni congiunzione And i sottoalberi non condividano variabili. Per avvicinarsi a questa proprietà il codice raggruppa i fattori di una congiunzione in componenti connesse per variabili. Se i gruppi sono indipendenti, si compone un And decomponibile; se invece la formula è intrecciata, si usa l'espansione di Shannon (shannon_expansion) scegliendo un pivot e riscrivendo f come $(s \wedge f|s=True) \vee (\neg s \wedge f|s=False)$.

Una volta garantita la decomponibilità, serve la deterministicità delle disgiunzioni per arrivare alla d-DNNF. Qui interviene DNNF2dDNNF che si appoggia a _make_or_deterministic: i rami di un Or devono essere mutuamente esclusivi. Se due rami si sovrappongono, si introduce un pivot e si applica di nuovo Shannon. È un modo costruttivo per imporre la determinism condition tipica della d-DNNF.

Il passaggio successivo rende la formula smooth (stessa frontiera di variabili per ciascun figlio di ogni Or), ottenendo la sd-DNNF. La funzione ddnnf2sdNNF fa questo introducendo nodi Tau(v). La classe Tau(Boolean) rappresenta la tautologia locale $(v \vee \neg v)$. Lo smoothing è importante perché consente di applicare in modo uniforme le regole di somma sui Or.

A pipeline definita, compile_to_sdDNNF orchestra le trasformazioni NNF $\rightarrow$ DNNF $\rightarrow$ d-DNNF $\rightarrow$ sd-DNNF. Il motivo teorico è che la knowledge compilation sposta la complessità nella fase di compilazione, per permettere di risolvere il conteggio modelli con una singola passata lineare.

Su questo circuito entra in gioco model_counting_sdnnf, implementazione del Weighted Model Counting. I letterali prendono pesi dal dizionario, Tau(v) contribuisce con w(v)+w(¬v), le congiunzioni moltiplicano, le disgiunzioni sommano. Con pesi unitari, la somma coincide con #SAT. exact_count_sdnnf compila in sd-DNNF, assegna pesi unitari e valuta il circuito.

Accanto alla via esatta, il notebook offre un canale approssimato. required_samples_eps_delta fornisce il numero di campioni necessari secondo Hoeffding, ma approximate_model_count usa un Monte Carlo con m=2000 campioni: genera assegnazioni random, stima la probabilità p■ di soddisfazione e calcola p■·2^n. Qui non si usa SampleSAT per stimare #SAT.

La funzione samplesat implementa un algoritmo di stochastic local search in stile WalkSAT. Lavora su CNF e alterna mosse random e greedy con probabilità di rumore. Se trova un'assegnazione soddisfacente la restituisce, altrimenti fallisce dopo i limiti di flip e riavvii. Serve quindi a trovare modelli, non a contarli.

Infine compare_exact_vs_approx costruisce un DataFrame che affianca exact e approximate count, n, numero di campioni ed errore relativo. style_pastel applica uno stile pastello alla tabella, render_kpi_cards genera card KPI con i valori principali, e render_pipeline mostra l'avanzamento della compilazione.

In sintesi, il notebook prende una formula proposizionale, la compila in sd-DNNF per ottenere un conteggio esatto dei modelli, affianca a questo un conteggio approssimato per campionamento uniforme, e fornisce strumenti sia algoritmici sia visivi per confrontare e spiegare i risultati.